

11

IBM

Systems Reference Library

IBM System/360 Basic Programming Support

FORTRAN IV, 360P-FO-031

Programmer's Guide

This publication describes the procedures for compiling and executing programs written in the Basic Programming Support FORTRAN IV language. It also includes the procedures for editing and updating the Basic Programming Support FORTRAN IV system and for preparing a Basic Programming Support FORTRAN IV job for a machine run.

PROPERTY OF
THE BROWN UNIVERSITY COMPUTING LABORATORY



PREFACE

This publication describes the procedures for (a) compiling and executing Basic Programming Support FORTRAN IV programs, (b) editing and updating the FORTRAN IV system, and (c) preparing FORTRAN IV jobs for machine runs.

This publication is divided into four major sections. The first section deals with compilation and execution. It covers the necessary control statements, the placement of those statements within an input deck, the minimum machine requirements for the various types of jobs, and the factors affecting the permissible size of source and object programs.

The second section deals with editing the compiler and/or updating the subprogram library. It covers the format of the FORTRAN system tape as supplied by IBM, and the control statements necessary to edit the system or update the system library.

The third section provides step-by-step operating instructions for running a job.

The fourth section includes appendixes describing compile- and object-time messages, information on writing subprograms in assembler language, FORTRAN-supplied subprograms, compiler output (object deck), and manually modifying the correspondences between data set reference numbers and actual devices.

FORTRAN programmers interested in sections one, two and four should be thoroughly familiar with the publication, Basic Programming Support FORTRAN IV Language, Form C28-6504. Systems programmers primarily interested in section two should be familiar with both the language publication and the publication, Basic Programming Support FORTRAN IV, Program Logic Manual, Form C28-6584. Operators interested in section three need nothing more than a general familiarity with the operation of System/360.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

INTRODUCTION	5	FORTRAN SYSTEM MAINTENANCE	27
Basic Programming Support FORTRAN IV System.	5	FORTRAN System Tape.	27
The FORTRAN Language.	5	Editing the FORTRAN System	28
Compilation and Execution	5	/EDIT Control Statement	28
System Maintenance.	5	Main Storage Size (xK)	28
Operating Procedures.	5	New System Tapes (n ₁ , n ₂ , n ₃ , . . .)	29
Subprogram Facilities	5	Format of /EDIT Statement.	29
Describing Control Statements.	5	/SET Control Statement.	29
COMPILATION AND EXECUTION.	7	12-2-9 REP Control Statement.	30
Control Statements	7	First-Byte Address	30
/Job Control Statement.	7	Segment Naming	30
SINGLE or MULTIPLE Option.	8	Object Code Replacement.	31
GO, GOGO, or NOGO Options.	8	Format of 12-2-9 REP Control Statement	31
Format of /JOB Statement	8	12-2-9 EDR Control Statement.	31
/FTC Control Statement.	9	First-Byte Address	32
NAME=xxxxxx Option	9	Segment Naming	32
DECK or NODECK Option.	9	Object Code Replacement.	32
LIST or NOLIST Option.	9	Format of 12-2-9 EDR Statement	32
MAP or NOMAP Option.	10	/AFTER Control Statement.	33
SIZE=xK Option	11	Library Subprogram Name.	33
Format of /FTC Statement	11	User-Written Subprogram Names.	34
/DATA Control Statement	12	Format of /AFTER Statement	34
Format of /DATA Statement.	12	/DELETE Control Statement	35
/SET Control Statement.	13	Names of Subprograms Deleted	35
Device Assignment Option	14	Format of /DELETE Statement.	35
Line=xxx Option.	15	/* (Asterisk) Control Statement	36
Format of /SET Statement	16	12-2-9 END Control Statement.	36
/Load Control Statement	16	Sample Edit Jobs.	36
MAP or NOMAP Option.	17	OPERATING INSTRUCTIONS FOR A FORTRAN JOB	39
TAPE or CARDS Option	17	Operator Messages.	39
Format of /LOAD Statement.	17	End of Job.	39
Deck Structure Using Control Statements.	17	Pause	39
Diagnostic Messages (Source Program)	20	Input/Output Interruption Messages.	39
Diagnostic Messages (Object Program)	20	FIS xxx.	39
Program Size Considerations.	20	FID xxx.	40
Source Program Size	20	FIA xxx.	40
Object Program Size	21	Machine Check Message	40
Compilation Speed Considerations	22	Types of Jobs.	40
Machine Considerations	22	Starting Instructions.	40
Minimum Machine Requirements.	22	APPENDIX A: SOURCE PROGRAM DIAGNOSTICS	42
1442 and 2540 Programming Considerations	22	APPENDIX B: OBJECT PROGRAM DIAGNOSTICS	50
The 1442 Card Read Punch	23	Error Code Diagnostics	50
The 2540 Card Read Punch	24	Program Interrupt Messages	51
Program Transition From 1442 to 2540.	24	Exponent-Overflow Exception.	52
		Exponent-Underflow Exception	52
		Floating-Point-Divide Exception.	52

2. Lower case letters and words are generic terms representing information that must be supplied; i.e., a substitution must be made when coding a parameter or option so represented.
3. Information within braces { } represents an option that may be included or omitted entirely, depending upon

the user's requirements. If one of the alternatives is underlined, the FORTRAN system assumes that alternative if the entire option is omitted.

4. An ellipsis (three periods) indicates that a variable number of items may be listed.

This section provides instructions for compiling and executing programs written in IBM System/360 Basic Programming Support FORTRAN IV Language.

The Basic Programming Support FORTRAN compiler accepts as input one or more source or object programs (or subprograms) in many combinations, called a FORTRAN "job," along with any associated input data.

In preparing a FORTRAN job, the user provides control information in control statements to tell the compiler how to handle a given type of job. To reduce the possibility of errors in using control statements, the compiler provides "standard options;" that is, in the absence of control information, the compiler makes assumptions on how to handle a given job.

The control statements, their placement within an input deck, diagnostic messages produced at source and object time, machine requirements, and program size considerations are described in the following text.

CONTROL STATEMENTS

Control statements are nonexecutable statements that inform the FORTRAN compiler about the nature of a job. There are two categories of control statements: compilation and execution statements; and FORTRAN system tape maintenance statements. The latter are described in the section "FORTRAN System Maintenance."

The compilation and execution control statements specify the following type of information:

1. The name of the job and/or the name of each individual program within it.
2. Whether the job consists of one program, or a combination of source and object programs.
3. Whether the programs within a job are to be:
 - a. Compiled only,
 - b. Compiled and executed, or
 - c. Executed only.

4. The type of output desired, such as program listing, object program card deck, or storage layout map.
5. The end of a particular job or the beginning of input data, if any.
6. The correspondence between data set reference numbers used in a program and the actual input/output (I/O) device addresses for a particular job.

Five control statements are used to specify the preceding information. They are: /JOB, /FTC, /DATA, /SET, and /LOAD. Each must be preceded by a slash (/).

With the exception of the /DATA control statement, all control statements are optional and have optional parameter lists. The absence of a particular parameter within a control statement, or in fact the statement itself, implies that a predefined specification (known as a standard option) is to be assumed by the compiler. Each control statement, along with its parameters, is described in the following text. Examples of control statements are in card format showing the column numbers in which the parameter list (if any) must begin.

/JOB CONTROL STATEMENT

The /JOB control statement is the first statement of a job. It has two purposes: (1) specifies whether the job consists of one or more compilations and (2) informs the FORTRAN compiler whether the job is to be compiled and executed or compiled only. (If the job consists solely of object programs, it may be executed by a /LOAD control statement; see the section, "/LOAD Control Statement.")

Associated with the /JOB control statement is a parameter list composed of two option classes:

1. SINGLE or MULTIPLE.
2. GO, GOGO, or NOGO.

SINGLE or MULTIPLE Option

The parameters SINGLE and MULTIPLE inform the FORTRAN compiler about the number of compilations to be performed for a certain job (i.e., one or several).

The parameter SINGLE specifies that the job consists of a single compilation of a source program or source subprogram. The job to which this parameter applies should not comprise more than one source program because only one compilation process is specified. However, as many object programs as desired may be included in the job.

The parameter MULTIPLE specifies that the job consists of several compilations, depending on the number of source programs and source subprograms in the job. The job itself may comprise as many object programs as desired because the parameter applies only to the number of compilations in a job; not the total number of programs in a job.

Embedded blanks are not permitted within the parameters. If neither SINGLE nor MULTIPLE is specified, the standard option is MULTIPLE.

GO, GOGO, or NOGO Options

The parameter GO specifies that the entire job is to be compiled and executed. If there are no serious source program errors, each program in the job (i.e., the compiled source programs and any object programs) is written on a tape (called a GO tape). When the entire job has been compiled and written on the GO tape, the tape is loaded into storage and execution commences. If, however, there are serious source program errors, execution is not attempted (see the section, "Diagnostic Messages").

The parameter GOGO also specifies that the job is to be compiled and executed. However, unlike the GO option, a GO tape is always produced and execution of the job is attempted regardless of any errors existing in the job. Note, however, that these errors may cause premature termination of the job.

The parameter NOGO specifies that the job is to be compiled only; no execution occurs and no GO tape is produced.

Embedded blanks are not permitted within the parameters. If neither GO, GOGO, nor NOGO is specified, the standard option is GO.

Format of /JOB Statement

Figure 1 shows the format of the /JOB statement.

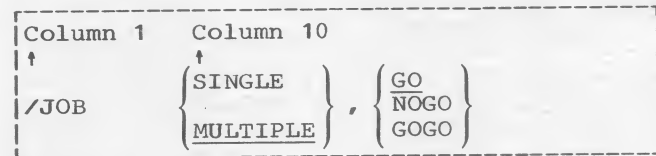


Figure 1. Format of /JOB Statement

The following are considerations in using the /JOB statement:

1. /JOB must start in column 1.
2. The option classes may appear in any order, the first starting in column 10.

Examples:

```
/JOB      SINGLE,NOGO
/JOB      GO
/JOB      GOGO,MULTIPLE
/JOB
```

3. Each parameter must be separated from the other only by a comma. The first blank encountered after column 10 results in whatever follows being regarded as comments. This comments field may be used for specifying the name of the job. The entire /JOB statement is always printed on-line.

Examples:

```
/JOB      SINGLE,GO PROBLEM1
/JOB      PROBLEM2
```

4. If two parameters in the same option class appear in a /JOB statement, the one appearing last is used.
5. If a /JOB statement is omitted from the job, the standard options assumed are:

```
/JOB      MULTIPLE,GO
```
6. Only one /JOB statement should be used for each job.

/FTC CONTROL STATEMENT

The /FTC control statement may precede each source program or source subprogram in a job. It has three purposes: (1) permits the user to uniquely name each program in his job, (2) requests optional output from a compilation such as object program on cards (deck), printout of source program, and a layout of storage reserved for program, and (3) allows the user to specify the size (in bytes) of main storage in which the job is compiled and/or executed.

Associated with the /FTC statement is a parameter list composed of five option classes:

1. NAME=xxxxxxx, where xxxxxxx is 1 through 6 characters, the first of which must be alphabetic.
2. DECK or NODECK
3. LIST or NOLIST
4. MAP or NOMAP
5. SIZE=xK, where x is in the range, $16 \leq x \leq 64$, and K represents 1024 bytes.

NAME=xxxxxxx Option

The parameter NAME=xxxxxxx specifies the name the user wishes to assign to the program or subprogram. If the job consists of more than one program, additional /FTC statements with different names may be used, one for each program. If this is done, each /FTC statement should precede each program, respectively. However, there is no necessity in specifying a name for each program in a job. If no specification is made, the assumed name for a main program is MAIN; for a subprogram the assumed name is the subprogram name in the header source statement (e.g., FUNCTION SUM(X,Y)).

Embedded blanks are not permitted within the parameter NAME=xxxxxxx.

DECK or NODECK Option

The parameter DECK specifies that the compiled source program (i.e., the object program) is to be punched on cards; the parameter NODECK specifies no object program deck output.

If the job is GO or GOGO and the DECK option is specified, the object program deck is produced in addition to the GO tape containing the object programs. These object programs produced on the GO tape are loaded into storage for execution; the object deck is not used. It is merely additional output requested by the programmer.

If the job is NOGO, the compiled source program can be executed only if the DECK option is specified. In this case, however, execution of the object deck is considered to be a different job. If it is desired to compile and execute all in one job, the GO or GOGO option should be specified.

Embedded blanks are not permitted within the parameters. If neither DECK nor NODECK is specified, the standard option is NODECK.

LIST or NOLIST Option

The parameter LIST specifies that the source program is to be printed (listed) on the device associated with data set reference number 3 (usually a printer); the parameter NOLIST specifies no listing.

Embedded blanks are not permitted within the parameters. If neither LIST nor NOLIST is specified, the standard option is LIST.

An example of a LIST printout is shown in Figure 2. This printout is the source program listing of Sample Program 1, shown in Appendix D, of the publication IBM System/360: Basic Programming Support FORTRAN IV, Form C28-6504.

```

BPS FORTRAN IVD COMPILER  VERSION 1  LEVEL 0   JUNE 1965
/FTC      DECK
BEGIN COMPILATION
-          C      PRIME NUMBER PROBLEM
- S.0001    100 WRITE (6,8)
- S.0002      8 FORMAT (52H FOLLOWING IS A LIST OF PRIME NUMBERS FROM 1 TO 1000/
-          119X,1H1/19X,1H2/19X,H3)
- S.0003    101 I=5
- S.0004      3 A=I
- S.0005    102 A=SQRT(A)
- S.0006    103 J=A
- S.0007    104 DO 1 K=3,J,2
- S.0008    105 L=I/K
- S.0009    106 IF (L*K-I) 1,2,4
- S.0010      1 CONTINUE
- S.0011    107 WRITE (6,5) I
- S.0012      5 FORMAT (I20)
- S.0013      2 I=I+2
- S.0014    108 IF (1000-I) 7,4,3
- S.0015      4 WRITE (6,9)
- S.0016      9 FORMAT (14H PROGRAM ERROR)
- S.0017      7 WRITE (6,6)
- S.0018      6 FORMAT (31H THIS IS THE END OF THE PROGRAM)
- S.0019      CALL DUMP(I,L)
- S.0020    109 STOP
- S.0021      END

END OF COMPILATION MAIN

```

Note: The numbers S.0001 through S.0021 represent sequential internal statement numbers which are assigned to each source statement by the compiler. Comments lines and continuation lines are not assigned internal statement numbers.

Figure 2. Sample Program Listing

MAP or NOMAP Option

The parameter MAP specifies that a storage map of the source and object program is to be printed on-line; the parameter NOMAP specifies no printout.

A storage map at compilation time gives a complete listing of:

1. The relative addresses and names of all variables used in the source program, including FUNCTION names, and in-line subprogram names.
2. The relative addresses and names of all external references made by the program, including all subprograms except in-line subprograms.
3. All user-specified literal constants in a program along with their relative addresses (e.g., in the statement $A=2.0$; 2.0 is a user-specified literal constant).
4. All compiler-generated constants along with their relative addresses.

5. A branch list consisting of all referenced statement numbers (except FORMAT statement numbers) in a program along with their relative addresses.
6. The relative address of the entry point where execution is to begin.
7. The size in bytes of the program.
8. The size in bytes of the blank COMMON area.

A storage map at execution time gives a complete listing of the absolute addresses (in hexadecimal) at which each object program in a job is loaded.

Embedded blanks within the MAP or NOMAP parameters are not permitted. If neither option is specified, the standard option is MAP.

An example of a MAP printout is shown in Figure 3. This printout is the source program map of Sample Program 1, shown in Appendix D, of the publication IBM System/360: Basic Programming Support FORTRAN IV, Form C28-6504.

STORAGE MAP VARIABLES (TAG C = COMMON, E = EQUIVALENCE)							
NAME TAG	REL ADR	NAME TAG	REL ADR	NAME TAG	REL ADR	NAME TAG	REL ADR
I	000090	A	000094	J	000098	K	00009C
L	0000A0						
EXTERNAL REFERENCES							
NAME	REL ADR	NAME	REL ADR	NAME	REL ADR	NAME	REL ADR
SQRT	0000A4	DUMP	0000A8				
CONSTANTS							
NAME	REL ADR	NAME	REL ADR	NAME	REL ADR	NAME	REL ADR
00000005	0000C4	00000002	0000C8	000003E8	0000CC		
IMPLIED EXTERNAL REFERENCES							
NAME	REL ADR	NAME	REL ADR	NAME	REL ADR	NAME	REL ADR
IBCOM	000168						
STATEMENT NO.	REL ADR	STATEMENT NO.	REL ADR	STATEMENT NO.	REL ADR	STATEMENT NO.	REL ADR
00100	00019E	00008	0000D0	00101	0001B4	00003	0001BC
00102	0001DC	00103	0001EA	00104	000208	00105	000210
00106	000220	00001	000238	00107	00024C	00005	00011C
00002	000268	00108	000274	00004	000288	00009	000120
00007	00029C	00006	000134	00109	0002BA		
S.0001	SIZE OF COMMON	00000					
	SIZE OF PROGRAM	00712					
<p>Note: The name and relative address of each statement number, constant, variable, and external reference is printed as shown. In addition, a tag of C or E is printed for each variable specifying whether it is in COMMON or EQUIVALENCE, respectively; otherwise, the tag field is left blank.</p>							

Figure 3. Sample Program Storage MAP

SIZE=xK Option

The parameter SIZE=xK, where x is in the range, $16 \leq x \leq 64$, specifies the size in bytes of usable main storage in which the source program is compiled. (K represents 1024 bytes.) The compilation speed of a job may be affected appreciably by the size of the machine. (In general, the larger the machine, the faster the compilation.)

If the specified SIZE option is greater than 64K (e.g., SIZE=256K), only the first 64K bytes of main storage are used for the

compilation. However, all main storage (e.g., 256K) is used for the execution.

Embedded blanks are not permitted within the SIZE=xK parameter. If no SIZE option is specified, the standard option is SIZE=16K.

Format of /FTC Statement

Figure 4 shows the format of the /FTC statement.

Column 1	Column 10
/FTC	$\left\{ \begin{array}{l} \text{NAME=xxxxxxx} \\ \text{NODECK} \end{array} \right\}, \left\{ \begin{array}{l} \text{DECK} \\ \text{NODECK} \end{array} \right\}, \left\{ \begin{array}{l} \text{LIST} \\ \text{NOLIST} \end{array} \right\}, \left\{ \begin{array}{l} \text{MAP} \\ \text{NOMAP} \end{array} \right\}, \left\{ \text{SIZE=xK} \right\}$

Figure 4. Format of /FTC Statement

The following are considerations in using the /FTC statement:

1. /FTC must start in column 1.
2. The parameters may appear in any order, the first starting in column 10.

Examples:

```
/FTC    NODECK,LIST,NOMAP,SIZE=32K
/FTC    NAME=PROB1,SIZE=64K,NODECK
/FTC
/FTC    LIST,MAP,NODECK,SIZE=16K
```

Note that the third and fourth examples above are effectively the same statements (see item 6).

3. Each parameter must be separated from the next only by a comma. The first blank encountered after column 10 results in whatever follows being regarded as comments. This comments field may be used for any purpose. The entire /FTC statement is printed along with comments whether or not the LIST or NOLIST option is specified.
4. If two parameters in the same option class appear in a /FTC statement, the one appearing last is used.
5. If a /FTC statement precedes only the first program in a job, all subsequent programs in the job are subject to the same specifications stated or implied in that statement. However, if an additional /FTC statement is encountered prior to any other program in a job, that program and all remaining programs (until the next /FTC statement appears) are subject to its specifications. For example, the following statements:

```
/FTC    NODECK,NOMAP,SIZE=32K
.
.
.
/FTC    NOLIST
.
.
.
```

results in the specification made in the first /FTC statement to hold for all programs up to the second /FTC statement. Thereafter, the second /FTC statement (along with its specified and standard options) is considered the specification for all remaining programs in the job.

6. At object time, the specifications in the last /FTC statement in the job take precedence. For example:

```
/FTC    SIZE=64K
.
.
.
/FTC    NAME=PROB2
.
.
.
```

The second /FTC statement would cause the SIZE option to be reset at 16K (the standard option). At object time the entire job would be executed assuming 16K of storage.

7. If no /FTC statements appear in a job, the standard options assumed for the entire job are:

```
/FTC    NODECK,LIST,MAP,SIZE=16K
```

/DATA CONTROL STATEMENT

The /DATA control statement is used to terminate all jobs. It should appear as the first card after the last source (or object) program in a job or immediately preceding any object time data cards.

If no /DATA control statement is specified, a warning message is produced and any cards remaining in the input device (e.g., card reader) are read until an end-of-file condition is sensed. Any data cards for use during execution may have been read and will have been lost for that job; however, normal operating procedure resumes.

Embedded blanks are not permitted within the characters /DATA. The /DATA control statement has no parameters. Comments within the statement are not permitted.

Format of /DATA Statement

Figure 5 shows the format of the /DATA statement.

```

+-----+
| Column 1 |
|   ↑     |
|  /DATA   |
+-----+
```

Figure 5. Format of /DATA Statement

The following are considerations in using the /DATA statement:

1. /DATA must start in column 1.
2. When using an IBM 1442 Card Read Punch in a compilation specifying DECK option, a sufficient number of blank cards for punching the object program must be inserted between the source program and the /DATA statement. An estimate should be made of the size of the object program to be punched on cards. (See the section, "Object Program Size.")
3. Only one /DATA statement should be used per job.

/SET CONTROL STATEMENT

The /SET control statement (1) specifies the number of print positions per line available on a particular printer and (2) changes the correspondence between data set reference numbers used in a program and the actual input/output device to which they refer. For example, a data set reference number may be associated with a particular tape device during execution of a program. Sometime later, during execution of a different program or re-execution of the same program, that same data set reference number may refer to a card reader.

The correspondences of data set reference numbers and device addresses are established in the Device Assignment Table (DAT). At the initiation of job processing, the DAT is loaded from the FORTRAN system tape into main storage and remains there until another job is processed. Figure 6 shows the pertinent information in the DAT as supplied by IBM.

For example, data set reference number 0 corresponds to the FORTRAN system tape contained on a 2400 series tape device with an actual address (in hexadecimal) of 180.

The first digit of the actual input/output device address represents the channel. There may be up to seven channels, represented in hexadecimal as 0 to 6, where 0 is the multiplexor channel. The next two digits represent the device address on a particular channel and may be in the range of 00 to FF.

The actual device address and type of device should always correspond; such a correspondence is dependent solely on the machine configuration of a particular installation. Thus, the /SET control statement permits the user to change only the correspondence of a data set reference number with a particular device.

Data Set Reference Number	Actual Device Address (Hexadecimal)	Type of Device
0	180	2400
1	00C	2540 (Read)
2	00D	2540 (Punch)
3	00E	1403 (Print)
4	183	2400
5	00C	2540 (Read)
6	00E	1403 (Print)
7	00D	2540 (Punch)
8	182	2400
9	282	2400
10	283	2400
11	183	2400
12	281	2400 (GO)
13 ¹	181	2400 (Wk.A)
14	280	2400 (Wk.B)
15	009	1052 (Type)

¹This data set reference number is also referred to by the loader as its work tape.

Figure 6. Device Assignment Table (DAT)

The /SET control statement provides the facility for changing the DAT during:

1. A compilation (source time).
2. An execution (object time).
3. An edit of the FORTRAN system tape. (See the section, "FORTRAN System Maintenance.")

If the /SET control statement is used during a compilation and/or an execution, the DAT is changed only temporarily; that is, the specifications made in the /SET control statement apply only to the program or programs which it precedes. Therefore, if the user finds the correspondences in the DAT suitable for his needs, no /SET control statement is needed.

Associated with the /SET control statement is a parameter list composed of two option classes:

1. Device Assignment Option.
2. LINE=xxx.

Device Assignment Option

The device assignment option specifies the relationship of data set reference numbers and actual input/output device addresses. The option is written as:

b=adr (tttt ds)

b
represents a data set reference number (e.g., 5).

adr
represents the actual input/output device address in hexadecimal that corresponds to b (e.g., 00C).

(tttt ds)
represents the type of device (e.g., 2540) with a fixed actual address of adr.

The tttt portion may specify the following type of I/O devices: 2400 Series tape devices, 2540 or 1442 Card Read Punch devices, 1403 or 1443 Printers, and the 1052 Printer-Keyboards. To specify a 7-track 2400 Series tape device, the tttt portion is written as 2400 followed by a ds portion.

The ds portion is specified only when an input/output device address (adr) refers to a 7-track tape. In this case, d designates the density in which data is read or written and s represents how the data is transferred. (If the ds is omitted, the standard option is a 9-track tape.)

The d may have one of three forms: L, M, or H.

1. L for low density of 200 characters per inch.
2. M for medium density of 556 characters per inch.
3. H for high density of 800 characters per inch.

The s may also have one of three forms: C, T, or N. However, the machine in which the job is processed must have the special feature of C and T data conversion. No special feature is required for N data conversion.

1. The C conversion is used to communicate with current systems when reading and writing binary data. As such, it enables the user to: (1) write data from System/360 storage (8-bit EBCDIC configuration) onto a 7-track tape without the loss of any bits; and (2) read data from a 7-track tape into System/360 storage, reorganizing the bits into 8-bit configuration. The data written on the 7-track tape is not the BCDIC equivalent of the EBCDIC data written from storage.

For example, Figure 7 shows the layout of three bytes (eight bits each) in storage which, when written, are organized as four 6-bit characters on a 7-track tape. Note that no bits are lost in the data transfer. The transformation of bits from storage to 7-track tape is a direct process; the transition stage is shown only to illustrate the type of change.

If the number of bytes written from storage does not produce an exact number of characters on a 7-track tape, the proper number of zeros are inserted to the right of the remaining bits producing an exact number of characters. For example, if four bytes are written from storage onto a 7-track tape, the data is organized as six characters; the last character consists of the last two bits of the fourth byte followed by four zeros. On input, the six characters are read into storage and reorganized as four bytes; the last four zeros are ignored.

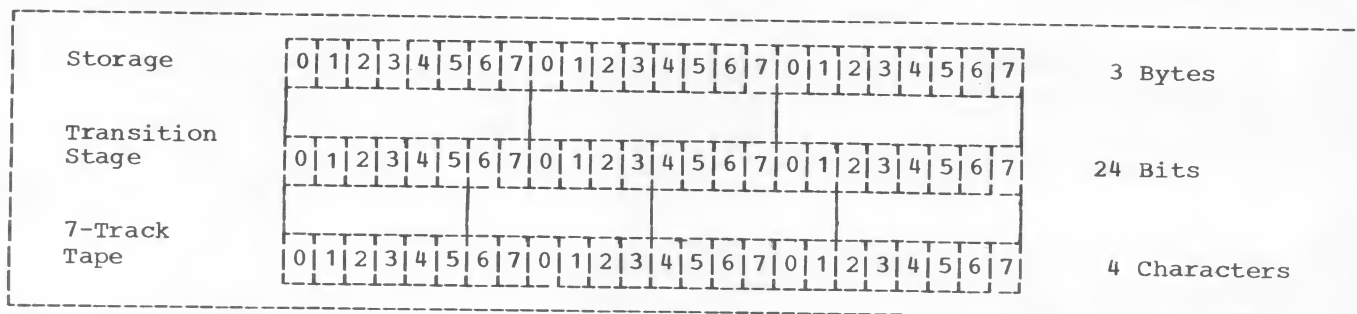


Figure 7. Data Transfer From Storage to a 7-Track Tape Using C Option

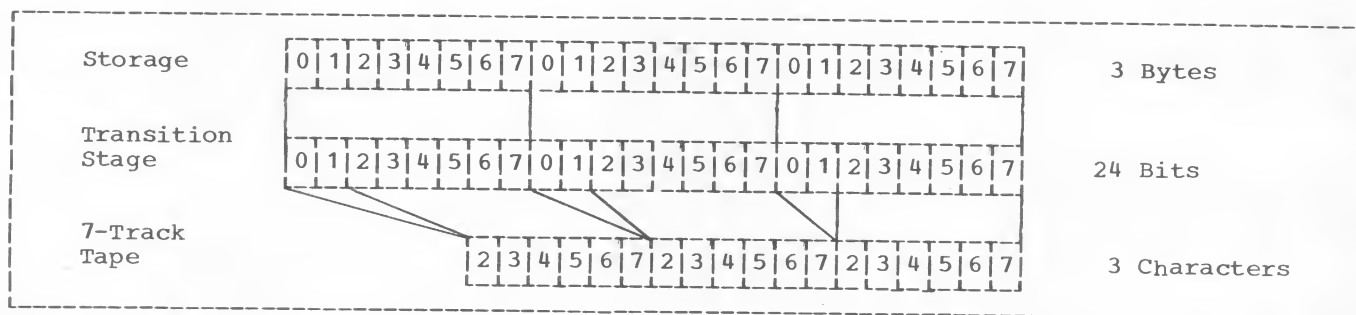


Figure 8. Data Transfer From Storage to a 7-Track Tape Using N Option

- The N conversion enables the user to:
 - (1) write data from System/360 storage (8-bit EBCDIC configuration) onto a 7-track tape such that the leftmost two bits of each byte are truncated;
 - (2) read data from a 7-track tape into System/360 storage, inserting two high-order zero bits for each 6 bits read, thereby producing an 8-bit configuration. For example, Figure 8 shows the layout of three bytes in storage, which, when written, are organized as three 6-bit characters on a 7-track tape. Note that the leftmost two bits of each byte are truncated in the process. The transformation of bits from storage to 7-track tape is a direct process; the transition stage is shown only to illustrate the type of change.

- The T conversion is used to communicate with current systems when reading and writing BCD data. As such, it enables the user to:
 - (1) write data from System/360 storage (8-bit EBCDIC configuration) onto a 7-track tape, converting each byte to its BCDIC equivalent;
 - (2) read data from a 7-track tape (6-bit BCDIC configuration) into System/360 storage, converting each character to its EBCDIC equivalent.

Embedded blanks are not permitted within the device assignment option. If the option is not specified, the correspondences in the DAT are the standard options. If the ds portion is omitted, the standard option is a 9-track tape.

Line=xxx Option

The LINE=xxx option, where xxx is a decimal number, specifies the permissible number of print positions per line on a printer. The xxx portion should not exceed the actual number of print positions available on the printer.

An incorrect format specification in a source program FORMAT statement might not be detected by the FORTRAN system (i.e., no error message is produced) if the xxx portion exceeds the actual number of print positions available. For example, assume that there are 132 print positions available on a printer associated with data set reference number 5. Then if a source program containing the statements:

```
10 FORMAT (5I30)
   .
   .
   .
WRITE (5, 10) I, J, K, L, M
```

were preceded by the specification LINE=150 in a /SET statement, the FORMAT statement would not be recognized as an error, because its specification is in agreement with that in the /SET statement. However, during object-time execution of the program, only the first 132 of the 150 specified characters are printed.

Embedded blanks are not permitted within the option. If LINE=xxx is not specified, the standard option is LINE=120.

Format of /SET Statement

Figure 9 shows the format of the /SET statement.

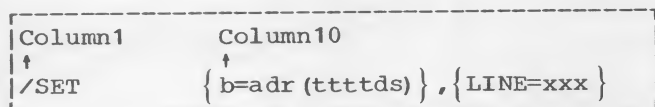


Figure 9. Format of /SET Statement

The following are considerations in using the /SET statement:

1. /SET must start in column 1.
2. The parameters may appear in any order, the first starting in column 10.

Examples:

```
/SET      3=280 (2400LT) ,LINE=132
/SET      LINE=144,2=00A (1442)
```

3. Each parameter must be separated from the next only by a comma. The first blank encountered results in whatever follows being regarded as comments.
4. If several LINE specifications are made or the same data set reference number is defined more than once in a /SET statement, the ones appearing last are used.
5. The specifications made in a /SET statement at compilation time apply to the program it precedes and all remaining programs in the job unless overridden by additional /SET statements. When a /SET statement is overridden by another /SET statement, the specifications made in the last /SET statement are used for the entire job at execution time. However, if a /SET statement defining a relationship between a data set reference number and a particular device is followed by

another /SET statement defining a different data set reference number, both specifications hold at compilation and execution time.

Example 1:

```
/SET      3=180 (2400)
.
.
.
program 1 statements
.
.
.
/SET      3=280 (2400)
.
.
.
program 2 statements
.
.
.
```

Explanation: The specification made preceding program 1 holds until overridden by the second /SET statement as shown. The second specification holds for program 2.

Example 2:

```
/SET      5=380 (2400)
.
.
.
program 1 statements
.
.
.
/SET      3=180 (2400)
.
.
.
program 2 statements
.
.
.
```

Explanation: The specification made preceding program 1 pertains to both program 1 and program 2. The specification made in the second /SET statement applies only to program 2.

6. If no /SET statements appear in a job, the specifications within the device assignment table hold. In addition, LINE=120 is the standard option.

/LOAD CONTROL STATEMENT

The /LOAD control statement immediately precedes the first object program in a job consisting of a single main object program

and several (if any) object subprograms. The /LOAD control statement is used to load all the object programs of a job into storage and transfer control for execution to the main program. In so doing, it: (1) permits the user to specify whether the job is to be loaded from tape or cards and (2) provides the option of printing the absolute load addresses (in hexadecimal) of each object program in the job.

Associated with the /LOAD control statement is a parameter list composed of two option classes:

1. MAP or NOMAP.
2. TAPE or CARDS.

MAP or NOMAP Option

The parameter MAP specifies that the absolute addresses (in hexadecimal), at which each object program in a job is loaded, are to be printed. The parameter NOMAP specifies no load addresses are printed.

If the job is both a compilation and an execution, the load address of each program is printed by specifying the MAP option in an /FTC control statement. (See the section, "/FTC Control Statement.")

Embedded blanks are not permitted within the MAP parameter. If neither MAP nor NOMAP is specified, the standard option is MAP.

TAPE or CARDS Option

The parameter TAPE specifies that the object programs in a job are to be loaded from the unit specified by data set reference number 12. (See the table of device address assignment in the section, "/SET Control Statement.")

The parameter CARDS specifies that the job is loaded from the unit specified by data set reference number 1.

Embedded blanks are not permitted within either parameter. If neither TAPE nor CARDS is specified, the standard option is CARDS.

Format of /LOAD Statement

Figure 10 shows the format of the /LOAD statement.

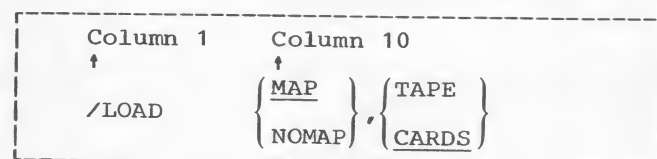


Figure 10. Format of /LOAD Statement

The following are considerations in using the /LOAD statement:

1. /LOAD must start in column 1.
2. The option classes may appear in any order, the first starting in column 10.

Examples:

```

/LOAD    CARDS,MAP
/LOAD    TAPE
/LOAD
  
```

Note that the first and third examples above are effectively the same statements.

3. Each parameter must be separated from the other only by a comma. The first blank encountered after column 10 results in whatever follows being regarded as comments.
4. If two parameters in the same option class appear in a /LOAD control statement, the one appearing last is used.
5. If the job is to be compiled and executed (i.e., GO or GOGO is specified), a /LOAD statement is not required. The compiler assumes that the compiled programs are to be loaded from tape.
6. If the job consists of several object main programs, the entire job is loaded into storage and control is passed to the last object main program loaded.

DECK STRUCTURE USING CONTROL STATEMENTS

The following deck structure examples illustrate how various job configurations may be combined with control statements to produce the desired results.

Example 1: Figure 11 shows a job consisting of one source program to be compiled and executed.

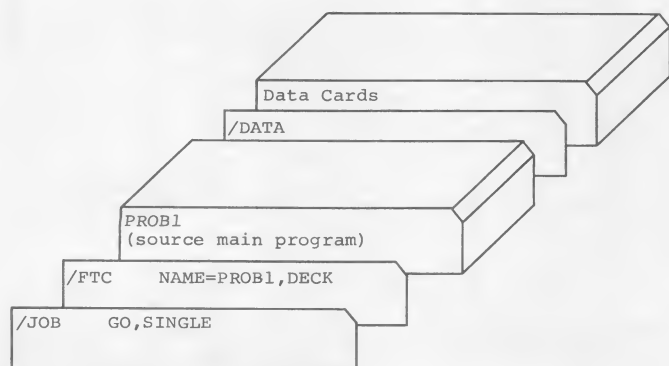


Figure 11. Example 1

The program, PROB1, is compiled (assuming a 16K machine) producing an object deck, listing of the source program, and a storage map. If no serious source program errors are encountered during compilation, the object program is loaded from the GO tape (a temporary storage tape containing the object program(s) produced by a GO or GOGO specification) into storage and executed. The data cards remain in the card reader until read in by the object program.

Example 2: Figure 12 shows a job consisting of one source main program and two source subprograms to be compiled and executed.

The program, PROB1, is compiled (assuming a 16K machine), producing a listing of the source program and a storage map. Execution of PROB1 does not commence until the remaining subprograms (SUB1 and SUB2) in the multiple job are compiled. Note that the second /FTC statement overrides the first /FTC statement preceding PROB1 such that neither a deck, list, nor storage map are produced when SUB1 and SUB2 are compiled.

Even if there are serious source program errors encountered when compiling the job, all three programs (PROB1, SUB1, and SUB2) are loaded from the GO tape into storage and control is passed to the main program PROB1. The data cards remain in the card reader until read by one or more of the object programs.

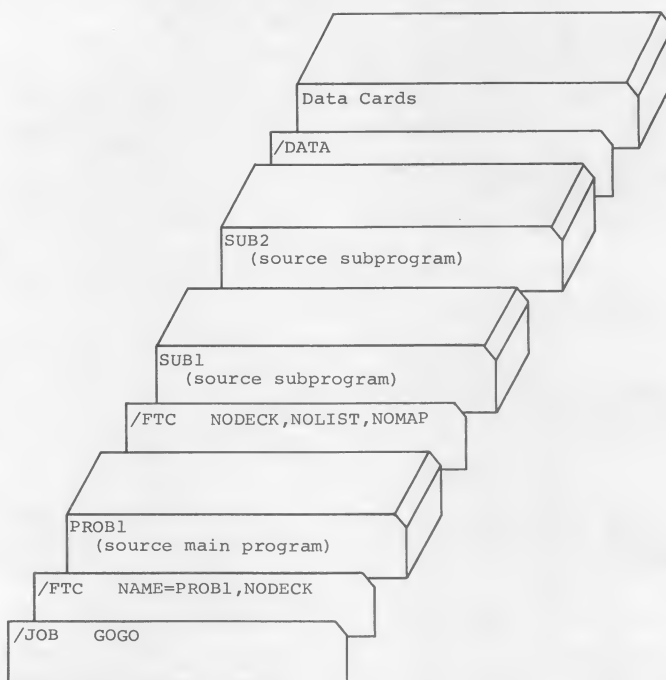


Figure 12. Example 2

Example 3: Figure 13 shows a job consisting of one source main program and one source subprogram to be compiled and executed according to changes made in the device assignment table.

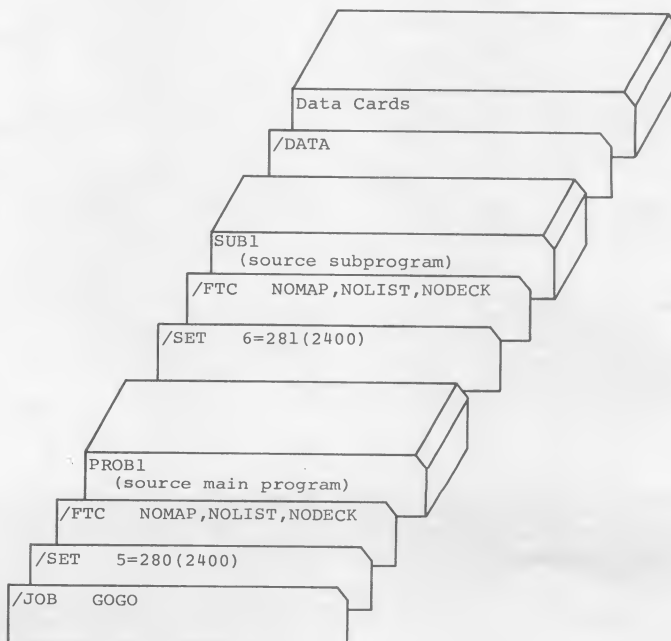


Figure 13. Example 3

The DAT is modified by the /SET statement following the /JOB statement PROB1 is then compiled (assuming a 16K machine) and placed on the GO tape. The next control statement (/SET) is read in again, modifying the DAT as shown; thereafter, subprogram SUB1 is compiled and placed on the GO tape.

Both object programs are loaded into storage and control is passed to PROB1. Use of data set reference number 5 in PROB1 refers to a 2400 series tape device whose actual address is 280; use of data set reference number 6 in subprogram SUB1 refers to a 2400 series tape with actual address of 281. The /SET statements in Example 3 result only in a temporary change to the DAT.

Example 4: Figure 14 shows a job consisting of one source main program, one object subprogram, and one source subprogram to be compiled and executed.

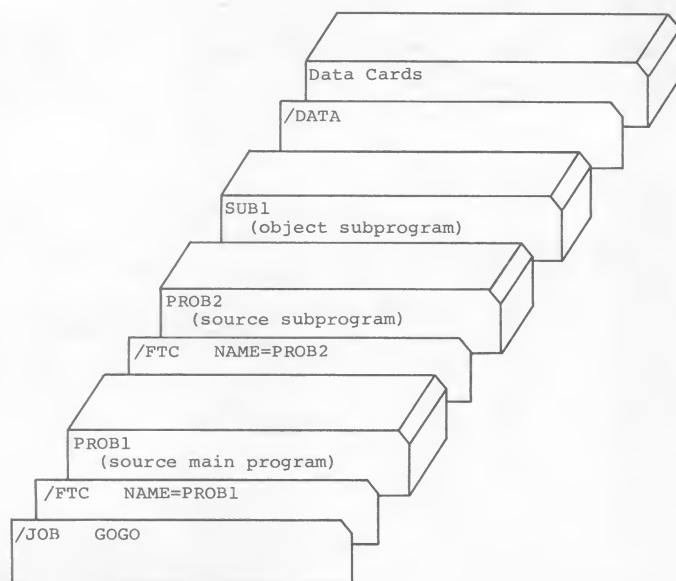


Figure 14. Example 4

The source program PROB1 is compiled producing a source program listing and storage map and placed on the GO tape.

The source subprogram PROB2 is then compiled producing a source program listing and storage map and placed on the GO tape. The object subprogram, SUB1, is read in and immediately placed on the GO tape. Note that it is not necessary for SUB1 to be preceded by any control statements. All three object programs are loaded into storage and control is transferred to PROB1.

Example 5: Figure 15 shows a job consisting of one source main program and one source subprogram to be compiled and executed using no control statements.

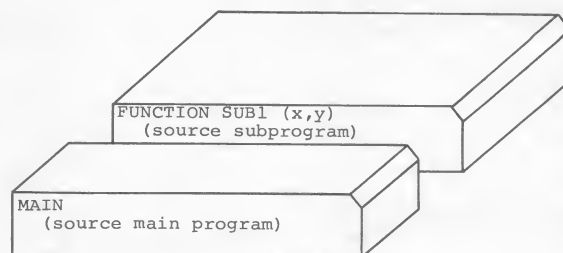


Figure 15. Example 5

Since there are no control statements, the assumed name of the main program is MAIN and for the subprogram SUB1; that is, the name specified in the header source statement of the subprogram.

Each program is compiled, placed on the GO tape if no serious errors are encountered, and loaded into storage. Control is transferred to MAIN. Note that data cards should not be placed behind SUB1 when the source programs are read in since a MULTIPLE job (standard option) is assumed and, in the absence of a /DATA statement, cards are read until an end-of-file condition is sensed. Thus, if there is input data, it should be read in from a device other than that used for reading in the source programs (e.g., a tape device or another card reader).

Example 6: Figure 16 shows a job consisting of one object main program and two object subprograms to be executed.

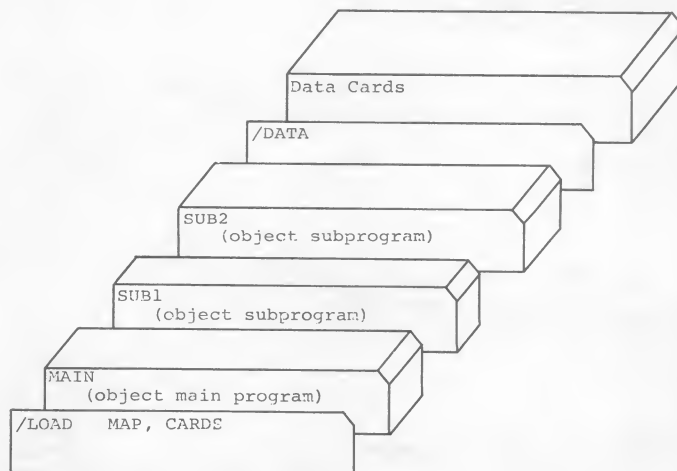


Figure 16. Example 6

The object main program, MAIN, along with the object subprograms, SUB1 and SUB2, are read from a card reader into storage. The absolute addresses (in hexadecimal) of each object program's position in storage are printed. Control is then transferred to MAIN.

Example 7: The following is a job consisting of two control cards with a source program on a tape whose address is 181.

```
/JOB      GOGO
/SET      1=181 (2400LC)
```

The /JOB and /SET statements are read in changing the DAT as shown. Because data set reference number 1 now refers to tape, the compiler reads in the source program from the tape, compiles it, and places the object program on the GO tape. The object program is loaded from the GO tape into storage and executed.

DIAGNOSTIC MESSAGES (SOURCE PROGRAM)

During compilation of a FORTRAN job, diagnostic messages are produced when certain statements do not conform to FORTRAN language specifications. These messages are produced whether or not the LIST option is specified.

For each such incorrect FORTRAN statement encountered, one or more messages are produced showing:

1. The internal statement number of the statement in error.
2. The message number.
3. The message itself describing the type of error.

Appendix A contains a complete listing of all source program diagnostic messages produced, along with a description of each message.

DIAGNOSTIC MESSAGES (OBJECT PROGRAM)

During execution of an object program, diagnostic error codes are printed (listed) on the device associated with data set reference number 3 (usually a printer) when certain errors are encountered. These error codes are produced irrespective of whether the LIST or NOLIST option is specified.

There are three types of object-time errors that may be detected by the FORTRAN system:

1. Errors found by the Relocating Loader during loading of object programs.
2. Errors found by the IBCOM routine when handling I/O operations for the object program.
3. Errors incurred when incorrect arguments are used for certain library subprograms.

For each object-time error encountered, an error code number is printed. Appendix B contains a complete listing of all object program diagnostic error codes, along with a description of each code.

PROGRAM SIZE CONSIDERATIONS

This section discusses the factors affecting source program size (number of source statements), object program size, amount of main storage available, and speed of compilation.

SOURCE PROGRAM SIZE

The maximum source program size depends upon: (1) the size of main storage available, (2) the number of statement numbers and symbols, such as subprogram, variable, and array names used in the source programs; and (3) the size of the dictionary reserved by the compiler. (See the publication, IBM System/360 Basic Programming Support FORTRAN IV, Program Logic Manual, Form C28-6584 for a detailed description of the dictionary.)

For a given main storage size, there is a corresponding dictionary size automatically reserved by the compiler. The compiler places information, called dictionary entries, about all statement numbers and symbols in the dictionary. If the number of required dictionary entries for a source program exceeds the space reserved for the dictionary, the programmer must either (1) divide the program into a main program and several subprograms, or (2) reduce the number and size of symbols and statement numbers in the program.

Figure 17 shows the relationship among main storage size (in bytes), dictionary size (in bytes), and average permissible source program size for a compilation

Main Storage (in bytes)	Dictionary Size (in bytes)	Average Source Program Size Permitted for a Compilation (number of statements)
16K*	4K	400
32K	8K	1000
48K	12K	1500
64K	16K	2000
* K represents 1024 bytes		

Figure 17. Compilation-Time Relationships

(number of source statements). The source program sizes shown are only an average and do not represent the maximum source program size permitted for all programs.

OBJECT PROGRAM SIZE

The amount of storage reserved for a compiled source program (i.e., the object program) depends upon: (1) the amount of main storage available; (2) the size of the FORTRAN System Director; (3) the size of the IBCOM routine; and (4) the size of each different out-of-line library subprogram referred to by the object program.

The FORTRAN System Director, occupying a maximum of 4K bytes of storage, resides in main storage during the compilation and execution of a program. At object time, its primary duty is the handling of all the input/output operations and interrupts for the object program.

The IBCOM routine, occupying a maximum of 5K bytes, handles all the input/output conversions for the object program (in conjunction with the FORTRAN System Director) and processing program arithmetic interruptions (e.g., overflow, underflow), etc.

At object time, any out-of-line subprograms referred to by the object program are read into main storage. The names of all IBM supplied subprograms and the size of each is given in Appendix C. (In-line subprograms referred to by a source program become part of the object program and therefore are a factor in determining overall object program size.)

Taking these considerations into account, the amount of storage available for an object program may be determined by subtracting from the main storage size, the size of the FORTRAN System Director, the IBCOM routine, and each different out-of-line subprogram referenced.

Figure 18 shows the relationship among main storage size (in bytes), IBCOM and FORTRAN System Director size (in bytes), storage size remaining for object program(s) (in bytes), and average permissible source program size for a compilation and execution (number of source statements).

Note that an object program may be executed in a machine size (main storage) different from that used for compiling the program.

Main Storage (in bytes)	Storage for IBCOM and FORTRAN System Director (in bytes)	Storage for Object Program (s)	Average Source Pro- gram Size Permitted for a Compilation and Execution (num- ber of statements)	Average Number of Object Program Cards Produced
16K*	9K	7K	150	100
32K	9K	23K	400	250
48K	9K	39K	700	430
64K	9K	55K	1000	610
* K represents 1024 bytes				

Figure 18. Execution-Time Relationships

COMPILATION SPEED CONSIDERATIONS

During a compilation, information about the program is passed between various phases of the compiler to produce an object program. This information may be transmitted to each phase by storing it either in a reserved portion of main storage (called a buffer area) or on an input/output device (e.g., tape). The amount of information transmitted is solely dependent upon the size of the source program. In addition, the size of the buffer area reserved is proportional to the amount of main storage available. Because this information can be accessed faster within main storage than from input/output devices, maximum compilation speed will be attained when few or no input/output operations occur.

The number of input/output operations can be appreciably reduced by decreasing the size of the source program and/or increasing the main storage size, thereby increasing the buffer area. When the main storage size is equal to 16K bytes, the buffer size reserved is 100 bytes.

In addition to the buffer size consideration, compilation time may be reduced by limiting the number of output options (e.g., DECK, LIST, MAP) in a job.

MACHINE CONSIDERATIONS

This section discusses the various machine configurations required by the FORTRAN system.

MINIMUM MACHINE REQUIREMENTS

The machine configuration needed by a particular installation may vary according to the type (e.g., compile only) and the amount of processing to be performed for a certain job. For instance, large jobs may require an additional amount of main storage and/or tape devices. However, irrespective of the type and size of job processing, all installations must have the following minimum machine configurations:

1. An IBM System/360 with 16,384 (16K) bytes of storage and the Scientific Instruction Set.
2. An IBM 1442 or 2540 Card Read Punch.
3. An IBM 1443 or 1403 Printer.

4. From three to ten IBM 2400 Series Magnetic Tape devices, depending on the type of job processing performed.

In addition to the above machine configuration, an IBM 1052 Printer-Keyboard may be used.

The following are the four types of job processing permitted by the FORTRAN system and the corresponding number of tape units required.

Compile Only (i.e., NOGO option): Three tape devices are required; two for work tapes and one for the FORTRAN system tape. If a large amount of main storage is available, two work tapes are not necessarily needed. The number of work tapes required (a maximum of two) depends on the amount of information about the program that is sequentially passed between each phase. If the program is small compared to the main storage size, little or no I/O operations occur; all the information is kept in buffer areas in main storage. Therefore, the number of work tapes may be reduced. (For a detailed discussion of buffer area reservation, see the section, "Compilation Speed.")

Compile and Execute (i.e., GO or GOGO options): Four tape devices are required; two for work tapes, one for the FORTRAN system tape, and one for the GO tape on which each compiled program is placed.

Execute Only (i.e., /LOAD statement): Two tape devices are required; one for the FORTRAN system tape and the other for a work tape (used by the loader). An additional tape unit may be required depending on whether the object programs are loaded from tape or cards (see the section, "/LOAD Statement").

FORTRAN System Tape Edit (i.e., /EDIT Statement): Two tape devices are required; one for the FORTRAN system tape, and the other for the new system tape generated by the edit. Up to eight additional tape devices may be required depending on the number of new system tapes to be created as specified in the /EDIT statement. (See the section, "FORTRAN System Maintenance.")

Figure 19 shows the number of tape devices required for each type of job.

1442 AND 2540 PROGRAMMING CONSIDERATIONS

This section discusses special programming considerations to be noted when using the IBM 1442 and 2540 Card Read Punch devices. It describes the manner in which

Type of Job Processing	FORTRAN System Tape	Work Tapes	New System Tapes	GO Tape	Total
Compile Only (NOGO)	1	2	0	0	3
Compile and Execute (GO or GOGO)	1	2	0	1	4
Execute Only (/LOAD)	1	1	0	1*	2 or 3
Edit (/EDIT)	1	0	1-9	0	2-10
*If object programs are on tape					

Figure 19. Tape Device Requirements

the 1442 and 2540 read and punch cards, the various data card deck structures permissible for each, and the actual source program I/O statements used to read and/or punch a card.

The 1442 Card Read Punch

The 1442 Card Read Punch has one input hopper in which data cards to be read and/or blank cards to be punched are placed. There are two areas, called stations, in which 1442 reading and punching operations are performed; one is the read station, the other a punch station. Every input card in the 1442 must pass both stations before being placed in the output hopper. (See Figure 20.)

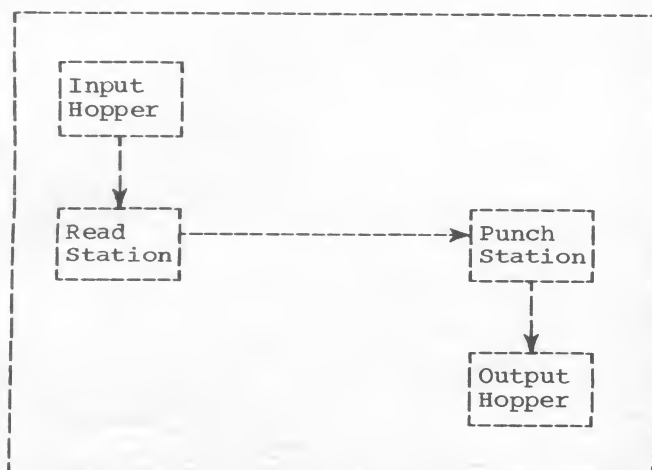


Figure 20. 1442 Card Read Punch Flow Diagram

Because every input card passes both stations, it is possible to read and punch the same card. This is done by following a FORTRAN source program READ statement with a WRITE statement.

For example, assume that data set reference number 5 refers to a 1442. The following FORTRAN statements would cause the second forty columns of a card to be read and the first forty columns of that same card to be punched in the specified manner:

```

.
.
.
10 FORMAT (T41,I10, F10.3, I5, F15.6)
11 FORMAT (4I10)
.
.
.
READ (5,10) I, A, J, B
WRITE (5,11) K, L, M, N
.
.
.

```

Instead of reading and punching the same card, it may be desirable to read a data card and punch a successive blank card. This may be done by following a READ statement with a "dummy" READ statement and a WRITE statement, in that order. The dummy READ statement (1) moves the card just read by the previous READ statement through the punch station into the output hopper and (2) moves another card from the input hopper, through the read station, to the punch station. A dummy READ statement may be written in two ways:

1. Using a source program READ statement with no list, e.g., READ (5,10).
2. Following the last format code in a FORMAT statement with a slash (/) character, e.g., 10 FORMAT (8I10/).

The following examples illustrate the FORTRAN statements that may be used to read a data card and punch a successive blank card. Note that the statements in Examples 1 and 2 perform the same function.

Example 1:

```

.
.
10 FORMAT (I20, F20.3, I10, F11.7, E19.6)
11 FORMAT (4I20)
.
.
READ (5,10) I, A, J, B, C
READ (5,10)
WRITE (5,11) K, L, M, N
.
.

```

Example 2:

```

.
.
10 FORMAT (I20, F20.3, I10, F11.7, E19.6/)
11 FORMAT (4I20)
.
.
READ (5,10) I, A, J, B, C
WRITE (5,11) K, L, M, N
.
.

```

A "dummy" READ statement is not needed if it is desired only to punch successive blank cards. For instance, the following example illustrates the FORTRAN statements that may be used to punch successive blank cards.

```

.
.
10 FORMAT (I20,F20.3,I10,F11.7,E19.6)
11 FORMAT (4I20)
.
.
WRITE (5,10) I,A,J,B,C
WRITE (5,11) K,L,M,N
.
.

```

The 2540 Card Read Punch

The 2540 Card Read Punch has two input hoppers: a read hopper and a punch hopper. Associated with the two hoppers are a read and a write station, respectively. Unlike the 1442, every input card in the 2540 passes only one of the two stations, depending on whether the card is in the read or write input hopper. (See Figure 21.)

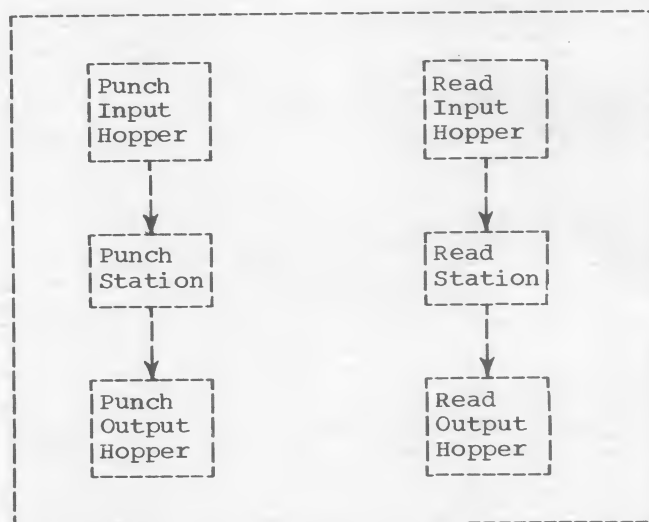


Figure 21. 2540 Card Read Punch Flow Diagram

Because every input card in the 2540 must pass either the read station or the punch station (depending on the input hopper in which the card is placed), it is not possible to read and punch on the same card. All data cards to be read should be placed in the read input hopper; all blank cards to be punched should be placed in the punch input hopper.

Program Transition From 1442 to 2540

If a system installation replaces a 1442 with a 2540, neither the I/O statements in a user-written program nor the data card deck structure for that program need be changed. However, it is important to note that the output deck structure may differ. For example, assume that data set reference number 5 refers to a 1442. Then the following FORTRAN statements would result in a data card being read and a successive blank card being punched.

Example:

```
.  
.  
.  
10  FORMAT (4I20/)  
11  FORMAT (4I20)  
.  
.  
.  
READ (5,10) I1, J1, K1, L1  
WRITE (5,11) I2, J2, K2, L2
```

The single output deck is ordered such that every input data card (the ones read) is followed by an output data card (the ones punched).

If data set reference number 5 referred to a 2540 in the preceding example, the statements would result in:

1. A data card being read.
2. A successive blank card (in the read input hopper) being passed to the output hopper.
3. A blank card (in the punch hopper) being punched.

There are two output decks. The deck in the read output hopper is ordered such that every input data card (i.e., the ones read) is followed by a blank card (i.e., the ones passed by the "dummy" READ). The deck in the write output hopper contains all the cards punched.

Note that the 2540, unlike the 1442, does not have the ability to read and punch on the same card. In this instance, the user must change his program.



The Basic Programming Support FORTRAN System may be tailored to fit the programming requirements of a particular installation. This section is directed to the system programmers responsible for expanding and maintaining (editing) the system. Before attempting any modification of the system, the user should have a thorough knowledge of the internal logic of the FORTRAN system.

A certain segment of the FORTRAN system, namely the editor, makes it possible for the user to revise, through the use of control statements, one or more portions of the system tape by adding, replacing, or deleting features to meet the requirements of his installation.

The format of the FORTRAN system tape supplied by IBM, including the order in which the library subprograms are supplied, and the control statements used to effect a change to the system are described in the following text.

FORTRAN SYSTEM TAPE

The FORTRAN system tape comprises the Initial Program Load, the FORTRAN System Director, the Control Card Routine, seven

phases of the compiler (Phases 10, 12, 14, 15, 20, 25, and 30), the Relocating Loader, the library, IBCOM, and the Editor. Each segment, except the library, consists of one record. The format of the library is in 80-character card images. Figure 22 shows the layout of the system tape as supplied by IBM.

To reduce access time when using library subprograms, they are placed on the system tape so that:

1. Subprograms that refer to other subprograms in the library for their calculation come first.
2. Subprograms that make no reference to other subprograms in the library come next.

The user should arrange any additional library routines according to the same principle. Figure 28 (see the section, "/AFTER Control Statement") shows the format of the library as supplied by IBM. Figure 23 shows examples of proper ordering when adding routines to the library. The arrow in these examples means that one routine refers to another.

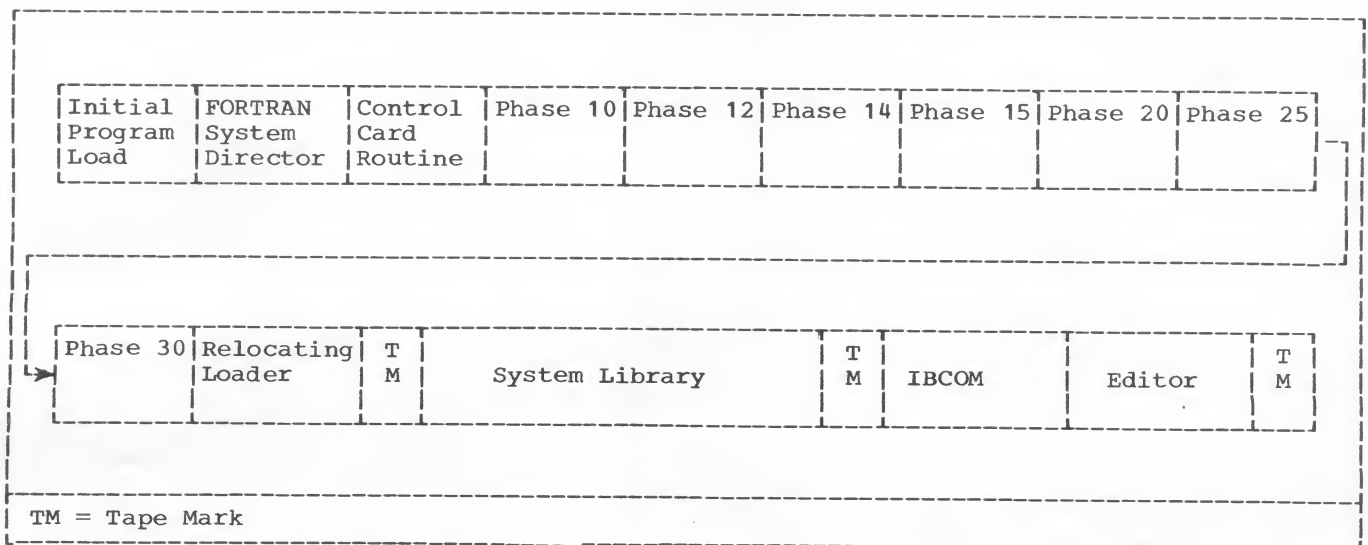


Figure 22. FORTRAN System Tape Layout

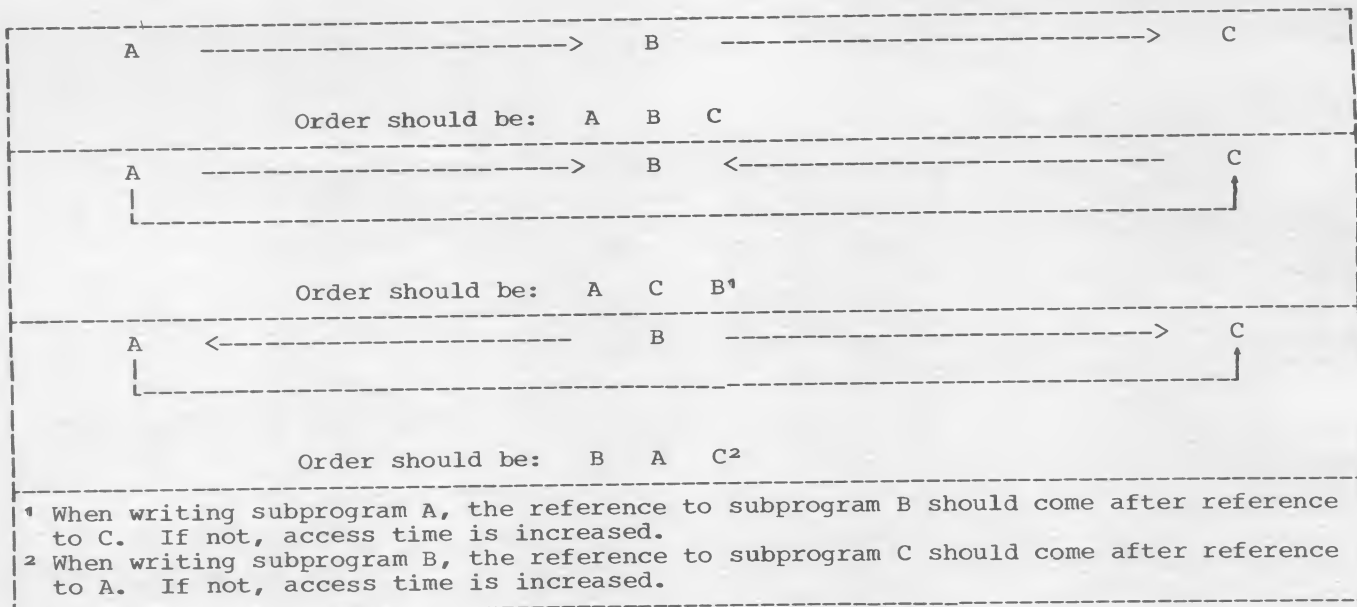


Figure 23. Ordering User-Written Subprograms in the Library

EDITING THE FORTRAN SYSTEM

The system maintenance control statements necessary for editing the FORTRAN system tape specify the following information:

1. The number of new system tapes to be generated.
2. The correspondence between data set reference numbers and actual input/output (I/O) device addresses.
3. The library subprograms to be added or deleted from the FORTRAN system tape.
4. A duplication of the present FORTRAN system tape.
5. The segment or segments of the compiler that are to be modified.

Eight control statements (/EDIT, /SET, 12-2-9 REP, 12-2-9 EDR, /AFTER, /DELETE, /*, and 12-2-9 END) are used to specify the preceding information. With the exception of the /* and 12-2-9 END control statements, all have parameter lists. Each control statement, along with its parameters, is described in the following text. Examples of control statements are in card format showing the column numbers in which the parameter list (if any) must begin.

/EDIT CONTROL STATEMENT

The /EDIT control statement must be the first statement of a system edit job. The /EDIT statement has three purposes: (1) effectively, causes the editor to be read from the system tape and control to be passed to it, (2) allows the user to specify the size in bytes of main storage that becomes the standard option on the new system tape(s), and (3) specifies the data set reference number of each new system tape to be created.

Associated with the /EDIT control statement is a parameter list composed of two classes:

1. The size of main storage designated by $\underline{x}K$ where \underline{x} is in the range, $16 \leq \underline{x} \leq 64$, and K represents 1024 bytes.
2. The new system tapes designated by $\underline{n}_1, \underline{n}_2, \underline{n}_3, \dots$ where each \underline{n} represents a data set reference number with which each new system tape (s) is associated.

Main Storage Size ($\underline{x}K$)

The parameter $\underline{x}K$, where \underline{x} is in the range, $16 \leq \underline{x} \leq 64$, specifies the size in bytes of main storage. (K represents 1024 bytes.) The parameter must start in column 10.

Embedded blanks are not permitted within the xK parameter. If no main storage size is specified, the standard option is whatever main storage size is specified in a previous /FTC control statement within the same job. (See the section, "SIZE=xK Option.") If no /FTC statement was used prior to the /EDIT statement, the main storage size assumed is dependent on the specification in the current system.

New System Tapes (n_1, n_2, n_3, \dots)

The parameter list n_1, n_2, n_3, \dots , where each n is a data set reference number, specifies the data set reference number of each new system tape created. Nine new system tapes can be generated in one job. The parameter list must start in column 15.

Embedded blanks are not permitted within the n_1, n_2, n_3, \dots parameter list. At least one n must be specified because at least one new system tape is created for every edit process. Each new system tape must be generated with a density of 800 characters per inch. Otherwise, it cannot be properly loaded.

Format of /EDIT Statement

Figure 24 shows the format of the /EDIT statement.

Column 1	Column 10	Column 15
↑	↑	↑
/EDIT	{xK}	{ n_1, n_2, n_3, \dots }

Figure 24. Format of /EDIT Statement

The following are considerations in using the /EDIT statement:

1. /EDIT must start in column 1.
2. Each data set reference number (i.e., n_1, n_2, n_3, \dots) must be separated from the next only by a comma. The first blank encountered after column 15 results in whatever follows being regarded as comments. This comments field may be used for any purpose.

Examples:

```
/EDIT 16K 11,12,13 EDIT1
/EDIT 32K 14 UPDATE1
```

3. The main storage size specified by xK applies to all source programs that succeed the edit program, unless overridden by a /FTC control statement appearing prior to those source programs.
4. Only one /EDIT statement should be used per job.

/SET CONTROL STATEMENT

The /SET control statement, when used in an edit job of the system tape, has effectively the same function as when used solely in a compilation and execution job. In both cases it (1) specifies the number of print positions per line available on a particular printer and (2) changes the correspondence between data set reference numbers used in a program and the actual input/output device to which they refer.

However, when the /SET statement appears after a /EDIT statement, the /SET control statement causes (1) a permanent change to the LINE=xxx standard option and (2) a permanent change to the Device Assignment Table (DAT) on the new system generated. All future jobs processed under control of the new FORTRAN system are subject to its specifications.

The parameters associated with the edit-job /SET statement (i.e., LINE=xxx and the Device Assignment Option) are the same as those used in the /SET statement for compilation and execution. The use of the /SET statement is fully described in the section, "Compilation and Execution." However, there are several additional considerations in properly using the /SET statement during an edit:

1. The /SET statement must be preceded (though not necessarily immediately) by an /EDIT control statement.
2. The old system tape used to generate the new system tape(s) remains unchanged at the conclusion of the edit job.
3. The standard option for the LINE=xxx option on the new system tape(s) is dependent upon the specification made in generating the new tapes. For example, the statements:

```
/EDIT 16K 12
/SET LINE=132
```

```
.
```

would result in the standard option for LINE to be changed from LINE=120 (the specification in the FORTRAN system tape supplied by IBM) to LINE=132 on the new system tape. All future jobs processed under control of the new FORTRAN system are subject to the specification LINE=132 unless temporarily overridden by a /SET statement within a particular job.

4. When a /SET statement appears before an /EDIT statement, the DAT is temporarily changed for the edit processing only. The new system tape generated by the edit is not affected by that /SET statement. For example:

```
/SET      0=280 (2400LT)
/EDIT     16K      12
.
.
.
```

The /SET statement would cause the DAT to be changed such that data set reference number 0 (which refers to the system tape) is associated with a 2400 Series tape device whose address is 280. This association holds only for the edit job. That is, the edit job edits the system tape associated with a data set reference number 0 (in this case, the tape at location 280). The new system tape is associated with data set reference number 12. Consider as another example the statement:

```
/EDIT     16K      12
.
.
.
```

The edit job edits the system tape associated with data set reference number 0 (in this case, the tape whose location is specified in the DAT of the system tape to be edited). The new system tape is associated with data set reference number 12.

12-2-9 REP CONTROL STATEMENT

The 12-2-9 REP control statement is used to replace a specified segment of the FORTRAN system with object code.

For every different segment modification desired, a 12-2-9 REP control statement is required. In addition, each modification of a different segment must be terminated by a 12-2-9 END statement. For example, to modify portions of both Phase 10 and Phase 20, two 12-2-9 END control statements are

required; one is used immediately following the editing statements for Phase 10, the other, following the editing statements for Phase 20. (See the section, "END Control Statement.")

Associated with the 12-2-9 REP control statement is a parameter list composed of three classes:

1. The address of the first byte in a segment where replacement of object code begins.
2. The name of a particular segment, designated by an abbreviated name, in which replacement of object code occurs.
3. The actual object code (in hexadecimal numbers) that replaces the existing code.

First-Byte Address

The address of the first byte in a segment where replacement of object code begins is specified starting in column 6 of the 12-2-9 REP statement. It is designated by six hexadecimal characters representing a 24-bit address. This address must be on a half-word boundary. If the address is not specified, it is assumed that the entire specified segment is to be replaced by object code. (See the section, "Object Code Replacement.")

Embedded blanks are not permitted within the address.

Segment Naming

Any segment shown in Figure 25 in the FORTRAN system may be modified by referring to its abbreviated name. This abbreviated name, along with the preceding slash, must be specified starting in column 13. Note that neither the IBCOM routine, the editor, nor the system library may be modified by a 12-2-9 REP statement. Modification of those segments may be effected by other control statements. (For modifying IBCOM or the editor see the section, "12-2-9 EDR Control Statement"; for the library, see, "/DELETE Control Statement," and "/AFTER Control Statement.")

Embedded blanks are not permitted within any of the segment names.

Segment Name	Abbreviated Name
Initial Program Load	/IPL
FORTRAN System Director	/FSD
Control Card Routine	/CTL
Phase 10	/P10
Phase 12	/P12
Phase 14	/P14
Phase 15	/P15
Phase 20	/P20
Phase 25	/P25
Phase 30	/P30
Relocating Loader	/LDR

Figure 25. Segment Name Abbreviations

Object Code Replacement

The object code (in hexadecimal numbers), which replaces the designated portion of a segment, may be specified either in the 12-2-9 REP statement itself or, if the space on the card is insufficient, in additional 12-2-9 REP statements. The object code is placed starting in column 21 and is written as two byte hexadecimal numbers, each two bytes separated by a comma. For example, the hexadecimal coding F024157AC035 should be written starting in column 21 as F024,157A,C035. Embedded blanks are not permitted within the object coding.

If an entire segment (e.g., Phase 10) is to be replaced, the 12-2-9 REP statement may be followed by an object deck produced from a user's source program compilation. The only parameter in the 12-2-9 REP statement should be the segment name. For example, the statement:

```
REP          /P10
```

followed by an object deck would cause all of Phase 10 to be replaced by the object deck following the 12-2-9 REP statement.

Format of 12-2-9 REP Control Statement

Figure 26 shows the format of the 12-2-9 REP statement.

Column 1 ↑	Column 6 ↑	Column 13 ↑	Column 21 ↑
12 2 9 REP	{address}	segment name	{object coding}

Figure 26. Format of 12-2-9 REP Control Statement

The following are considerations in using the 12-2-9 REP statement:

1. 12-2-9 REP must be placed starting in column 1.
2. The first blank encountered in or after column 21 results in whatever follows being regarded as comments.
3. As many 12-2-9 REP statements as needed may be used to modify a segment. However, the last 12-2-9 REP statement used in modifying a particular segment must be followed by a 12-2-9 END control statement.

Example: (The 12-2-9 punch is not shown in examples.)

```

.
.
.
REP          002A02 /P10      F01A,235A,B302
REP          002A46 /P10      F20B,9E03
END          END OF PHASE 10 EDIT PROCESS
REP          001B04 /P20      2113,57EA
END          END OF PHASE 20 EDIT PROCESS
.
.
.

```

Explanation: The first 12-2-9 REP statement causes the specified code to replace existing code in Phase 10 from location 002A02 through location 002A07. The next 12-2-9 REP statement causes the specified code to replace the code from location 002A46 through location 002A49. The 12-2-9 END statement signals the termination of editing in Phase 10. The next 12-2-9 REP statement then causes the existing code in Phase 20 from location 001B04 through 001B07 to be replaced by 211357EA. The 12-2-9 END statement signals the termination of editing in Phase 20.

12-2-9 EDR CONTROL STATEMENT

The 12-2-9 EDR control statement is equivalent to the 12-2-9 REP control statement in that it is used to replace a specified segment of the FORTRAN system with object code. However, the 12-2-9 EDR control statement may be used to modify only the IBCOM routine or the editor.

When modification to IBCOM or the editor is specified by one or more 12-2-9 EDR control statements, those 12-2-9 EDR control statements must be followed by a 12-2-9 END control statement. Because the editor is at the end of the FORTRAN system tape, it is the last segment that can be modified; therefore, the 12-2-9 END control statement must be followed by a /* control statement. (See the sections, "/* (Asterisk) Control Statement" and "12-2-9 END Control Statement.")

Associated with the 12-2-9 EDR control statement is a parameter list composed of three classes:

1. The address of the first byte in IBCOM or the editor where replacement of object code begins.
2. The name of a particular segment, designated by an abbreviated name, in which replacement of object code occurs.
3. The actual object code (in hexadecimal numbers) that replaces the existing code.

First-Byte Address

The address of the first byte in a segment where replacement of object code begins is specified starting in column 6 of the 12-2-9 EDR statement. It is designated by six hexadecimal characters representing a 24-bit address. This address must be on a half-word boundary. If the address is not specified, it is assumed that the entire specified segment is to be replaced by object code. (See the section, "Object Code Replacement.")

Embedded blanks are not permitted within the address.

Segment Naming

Either the IBCOM routine or the editor in the FORTRAN system may be modified by referring to the abbreviated segment names: /IBC and /EDR, respectively. These abbreviated names, along with the preceding slash, must be specified starting in column

13. (To modify anything else, except the library subprograms, the 12-2-9 REP control statement is used. To modify the library, /AFTER and /DELETE control statements are used. (See the sections, "12-2-9 REP Control Statement," "/AFTER Control Statement," and "/DELETE Control Statement.")

Object Code Replacement

The object code (in hexadecimal numbers), which replaces the designated portion of IBCOM or the editor, may be specified either in the 12-2-9 EDR statement itself or, if the space on the card is insufficient, in additional 12-2-9 EDR statements. The object code is placed starting in column 21 and is written as two byte hexadecimal numbers, each two bytes separated by a comma. For example, the hexadecimal coding F024157AC035 should be written starting in column 21 as F024,157A,C035. Embedded blanks are not permitted within the object coding.

If the entire IBCOM routine and editor are to be replaced, the 12-2-9 EDR statement may be followed by an object deck produced from a user's source program compilation. The only parameter in the 12-2-9 EDR statement should be the abbreviated segment name /EDR or /IBC. For example, the statement:

EDR /EDR

followed by an object deck would cause the entire editor to be replaced by the object deck following the 12-2-9 EDR statement. The statement:

EDR /IBC

followed by an object deck would cause the entire IBCOM routine to be replaced by the object deck following the 12-2-9 EDR statement.

Format of 12-2-9 EDR Statement

Figure 27 shows the format of the 12-2-9 EDR statement.

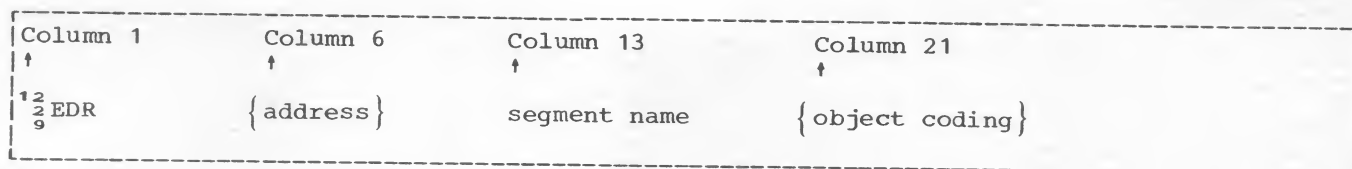


Figure 27. Format of 12-2-9 EDR Statement

The following are considerations in using the EDR statement:

1. 12-2-9 EDR must start in column 1.
2. The first blank encountered in or after column 21 results in whatever follows being regarded as comments.
3. As many 12-2-9 EDR statements as needed may be used to modify a segment. However, the last 12-2-9 EDR statement used in modifying a particular segment must be followed by a 12-2-9 END statement.

Example: (The 12-2-9 punch is not shown in examples.)

```

.
.
.
EDR    002A02 /EDR    F01A,235A,B302
EDR    002A46 /EDR    F20B,9E03
EDR    002B04 /EDR    2113,57EA
END    END OF EDITOR EDIT PROCESS
.
.
.

```

Explanation: The first 12-2-9 EDR statement causes the specified code to replace existing code in the editor from location 002A02 through location 002A07. The next 12-2-9 EDR statement causes the specified code to replace the code from location 002A46 through location 002A49. The next 12-2-9 EDR statement then causes the existing code in the editor from location 002B04 through 002B07 to be replaced by 211357EA. The 12-2-9 END statement signals the termination of editing for the editor.

/AFTER CONTROL STATEMENT

The /AFTER control statement is used to insert user-written subprograms in the system library. These user-written subprograms must be in object deck format (i.e., object coding) produced from either a Basic Programming Support assembler source program assembly or a FORTRAN source program compilation.

Associated with the /AFTER control statement are two parameter classes:

1. Library subprogram name after which the user-written subprogram(s) is inserted.
2. User-written subprogram names that are to be incorporated in the system library.

Library Subprogram Name

Associated with every library subprogram is a subprogram name and one or more entry names representing particular entry points in the subprogram. The entry points designate a particular portion of the subprogram that performs a certain function. Only entry points should be referred to in a FORTRAN source program. For example, associated with the library subprogram named SIN are two entry points: SIN and COS. Both of these entry points may be used in a FORTRAN source program to compute the sine and cosine of a given argument, respectively. However, only subprogram names may be used in an /AFTER statement to specify the library subprogram after which a user-written subprogram(s) is to be inserted.

In some instances, the subprogram name and entry point for a particular library subprogram are identical. For example, the name of the library subprogram AINT represents not only the subprogram name but also the entry point for that subprogram. The name AINT may be used in a source program to truncate the fractional portion of an argument. This name may also be used in a /AFTER statement to designate the subprogram after which a specified user-written subprogram(s) is inserted. Figure 28 shows the complete list of library subprogram names and associated entry points (shown indented) supplied by IBM, in the order in which they appear on the FORTRAN system tape.

The library subprogram name must be placed starting in column 10. Embedded blanks within the name are not permitted.

FDXPD ¹	DUMP	MAX1
FDXPI ¹	PDUMP	MIN1
FIXDI ¹	DLOG	AMAX1
FRXPI ¹	DLOG10	AMIN1
FRXPR ¹	DSQRT	DMAX1
EXIT	DATAN	DMIN1
IBERR	DTANH	FLOAT
CGOTO	DCOS	DFLOAT
ALOG	DSIN	IFIX
ALOG10	DEXP	INT
SQRT	MOD	IDINT
ATAN	AMOD	AIN1
TANH	DMOD	SLITE
EXP	MAX0	SLITET
COS	MIN0	OVERFL
SIN	AMAX0	DVCHK
	AMIN0	

¹Used at object-time to perform exponential operations.

NOTE: All indented names represent entry points. All other names are subprogram names as well as entry points.

Figure 28. Library Subprograms Supplied by IBM

User-Written Subprogram Names

The user-written subprogram names are the names of the subprograms in object deck format to be inserted in the library. The name of the subprogram is the name specified in the /FTC statement (if any) used in compiling the subprogram. If no /FTC statement was used, the name of the subprogram is the name specified in the header source statement (e.g., for the statement SUBROUTINE HOLD(X,Y), HOLD is considered the subprogram name).

The user-written subprogram names are placed starting in column 17, each separated by a comma. Embedded blanks are not permitted within the list of names.

Format of /AFTER Statement

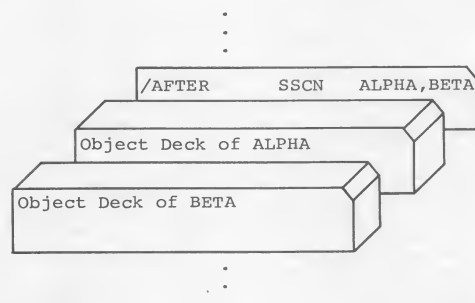
Figure 29 shows the format of the /AFTER statement.

Column 1	Column 10	Column 17
/AFTER	name	sub ₁ ,sub ₂ ,...,sub _n

Figure 29. Format of /AFTER Statement

The following are considerations in using the /AFTER statement:

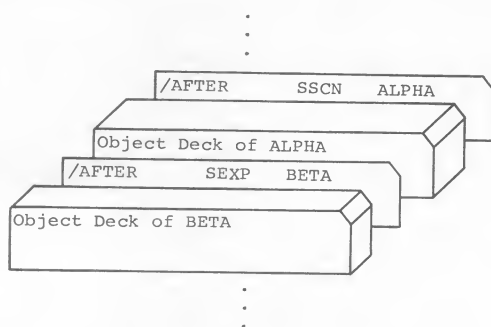
1. /AFTER must start in column 1.
2. The first blank encountered in or after column 17 results in whatever follows being regarded as comments.
3. The object subprogram deck that is to be inserted after a given library subprogram must be placed immediately following the /AFTER statement to which it corresponds. In addition, if a /AFTER statement specifies that several object subprograms are to be inserted after a given library subprogram, the object decks should appear in the same order in which their names are specified in the /AFTER statement.



Explanation: The user-written subprograms ALPHA and BETA are inserted in the system library after the subprogram COS.

4. As many /AFTER statements as needed may be used to insert subprograms in the library. However, the /AFTER statements should be arranged so that the library subprogram names are in the same order that the subprograms exist in the library.

Example: Assume that the subprogram EXP is preceded by COS in the library and that ALPHA and BETA are user-written subprograms.



Explanation: The preceding statements cause the user-written subprogram ALPHA (i.e., the object deck) to be inserted after the COS subprogram in the system library. The object deck of BETA is then inserted after the EXP subprogram. The /AFTER statements (along with the associated object decks) are ordered as shown because subprogram COS precedes EXP in the system library.

5. In an edit job, all /AFTER statements must precede any 12-2-9 EDR statements and follow any 12-2-9 REP statements.

/DELETE CONTROL STATEMENT

The /DELETE control statement is used to delete any specified library subprogram(s) from the system library.

Associated with the /DELETE control statement is a single parameter class consisting of the names of the library subprograms to be deleted.

Names of Subprograms Deleted

The names of the subprograms to be deleted must be placed starting in column 10, each separated by a comma. A complete list of all library subprogram names supplied by IBM along with the order in which they appear on the FORTRAN system tape is shown in Figure 28.

Format of /DELETE Statement

Figure 30 shows the format of the /DELETE statement.

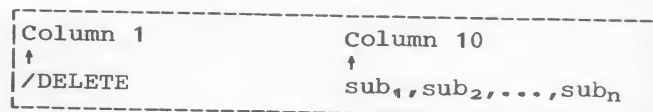
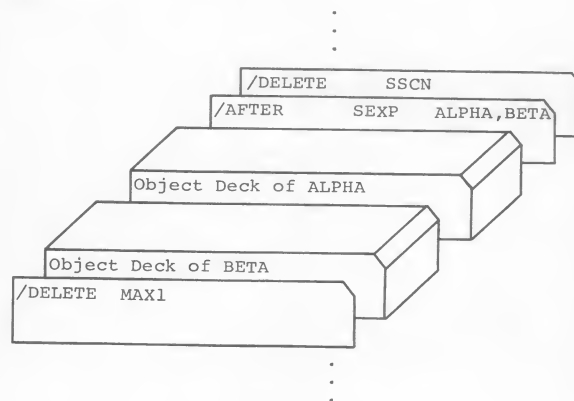


Figure 30. Format of /DELETE Statement

The following are considerations in using the /DELETE statements:

1. /DELETE must start in column 1.
2. The first blank encountered in or after column 10 results in whatever follows being regarded as comments.
3. /DELETE statements may be interspersed with /AFTER control statements. However, the /DELETE and /AFTER statements should be ordered so that the library subprograms to which they refer correspond to the sequence of the subprograms in the library.

Example: Assume the subprograms COS, EXP, and MAX1 are library subprograms in that order and ALPHA and BETA are user-written subprograms.



Explanation: The COS subprogram is deleted from the library. Because the SIN function is part of the subprogram COS, it also is deleted. The user-written subprograms ALPHA and BETA are inserted after the EXP subprogram and then MAX1 is deleted.

4. All /DELETE statements must appear after any 12-2-9 REP statements, but must precede any 12-2-9 EDR statements.

/* (ASTERISK) CONTROL STATEMENT

The /* control statement is the last statement in an edit job. It causes any remaining segments of the old system tape that were not edited to be written on the new system tape(s).

There are no parameters associated with the /* control statement. If a /EDIT and /* control statements are the only statements in an edit job, the old system tape is merely copied (written) on the new system tape(s).

The /* must appear starting in column 1. The first blank encountered after column 2 results in whatever follows being regarded as comments.

Example:

```
Column 1
↑
/*      THIS IS END OF EDIT JOB
```

12-2-9 END CONTROL STATEMENT

The 12-2-9 END control statement must be used to terminate an editing process for each segment specified in a 12-2-9 REP or 12-2-9 EDR control statement unless the control statements are followed by an object deck(s). In this case, the 12-2-9 END statement is not required since it is already a part of the object deck(s). The 12-2-9 END is placed starting in column 1. The first blank encountered after column 4 results in whatever follows being regarded as comments. Figure 31 shows an example of edit job using a 12-2-9 END control statement.

```

/EDIT  16K          8
      REP  002A02    /P10    C23A
      END  THIS IS END OF EDIT FOR PHASE 10
      EDR  002A46    /EDR    F20B,9E03
      END  THIS IS END OF EDIT FOR THE EDITOR
/*      REST OF TAPE COPIED ON NEW SYSTEM

```

Figure 31. Sample Edit Job

SAMPLE EDIT JOBS

Example 1: Figure 32 shows a card deck comprising a sample edit job.

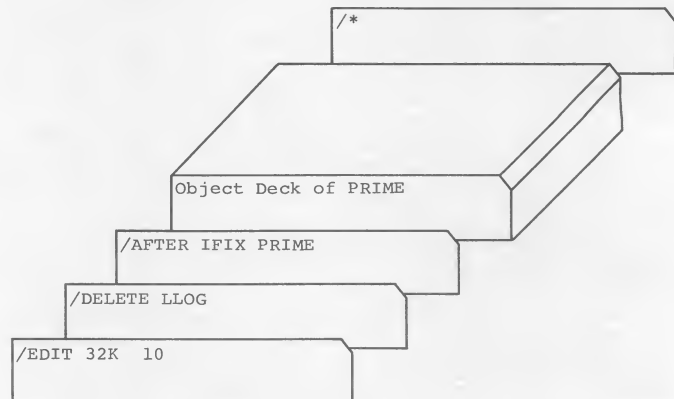


Figure 32. Sample Edit Job 1

The following is a step-by-step description of sample job 1 as it is processed:

1. After the machine operator presses the Load Key, the initial program load, FORTRAN system director, and control card routine are read into main storage. The first statement (i.e., the /EDIT statement) is read causing the editor to be read in from the system tape and main storage size to be set to 32K. Data set reference number 10 is specified as the tape on which the new system is generated. The initial program load is then written on the new system tape and the old system tape is rewound.
2. The next control statement (i.e., the /DELETE statement) is read in. Because it refers to the system library, all segments of the system prior to the library (i.e., FORTRAN system director, phases 10 through 30, and the relocating loader) are written on the new system tape followed by a tape mark.
3. The subprogram ALOG is then deleted from the system library.
4. The next control statement (i.e., the /AFTER statement) is read in. The object deck of the subprogram PRIME is inserted after subprogram IFIX.

5. The next control statement (i.e., the `/*` statement) is read in causing the remaining portion of the old system tape to be copied onto the new system tape. Both tapes are then rewound.

Example 2: Figure 33 shows a card deck comprising a sample edit job.

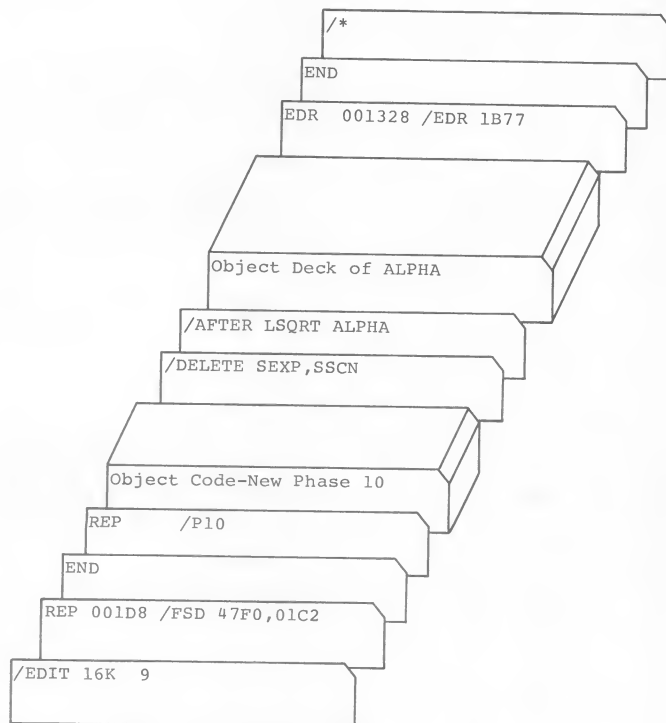


Figure 33. Sample Edit Job 2

The following is a step-by-step description of sample job 2 as it is processed:

1. After the machine operator presses the Load Key, the initial program load, FORTRAN system director, and control card routine are read into main storage. The first statement (i.e., the `/EDIT` statement) is read causing the editor to be read in from the system tape and main storage size to be set to 16K. Data set reference number 9 is specified as the tape on which the new system is generated. The initial program load is then written on the new system tape and the old system tape is rewound.
2. The 12-2-9 `REP` statement causes the object code 47F001C2 to replace the existing code in the FORTRAN system director from location 0001D8 through 001DB.

3. The 12-2-9 `END` statement specifies that editing for the FORTRAN system director is complete. The modified FORTRAN system director is then written on the new system tape.
4. The next 12-2-9 `REP` statement is read in. Because it specifies that all of phase 10 is to be replaced by an object deck following the card, the control card routine (which precedes phase 10) is written on the new system tape. Then phase 10 is read from the old system tape into main storage and completely replaced by the object deck following the 12-2-9 `REP` statement.
5. The last statement in the object deck of phase 10 (i.e., the 12-2-9 `END` statement) specifies that editing for phase 10 is complete. The new phase 10 is then written on the new system tape.
6. The next control statement (i.e., the `/DELETE` statement) is read in. Because it refers to the system library, all segments of the compiler prior to the library (i.e., phases 12, 14, 15, 20, 25, 30 and the relocating loader) are written on the new system tape followed by a tape mark.
7. The subprograms `EXP` and `COS` are then deleted from the system library.
8. The next control statement (i.e., the `/AFTER` statement) is read in. The object deck of the subprogram `ALPHA` is inserted after subprogram `SQRT`.
9. The next control statement (i.e., the 12-2-9 `EDR` statement) is read in. Because no more modification of the library is to be made, any remaining subprograms in the library are written on the new system tape and a tape mark is written and `IBCOM` is copied.
10. The 12-2-9 `EDR` statement then causes the object code 1B77 to replace the existing code in the editor from locations 001328 through 001329.
11. The 12-2-9 `END` statement specifies that editing for the editor is complete. The modified editor is then written on the new system tape followed by a tape mark.
12. The next control statement (i.e., the `/*` statement) is read in causing the remaining portion of the old system tape to be copied onto the new system tape. Both tapes are then rewound.



This section describes the messages which signal the need for operator intervention and the procedures an operator should follow in preparing a FORTRAN job for a machine run.

The only prerequisite for setting up a FORTRAN job is a familiarity with System/360 operation.

OPERATOR MESSAGES

There are only three types of messages produced by the FORTRAN system requiring operator intervention. They are:

1. END OF JOB.
2. PAUSE xxxxx, where xxxxx is a one to five digit number.
3. Input/Output interruption message: FIS xxx, FID xxx, and FIA xxx, where xxx is the actual address of an I/O device.
4. Machine check message FMS, displayed in hexadecimal as C6D4E2 in the low-order portion of the PSW.

END OF JOB

An END OF JOB message is printed when the job has completed its specified processing. For example, if the job is to be compiled and executed, the END OF JOB message is printed at the conclusion of the execution process.

When the END OF JOB message is printed, the operator can prepare the machine for the next job.

PAUSE

A PAUSE xxxxx message is printed when a job being executed encounters a compiled PAUSE statement.

If a PAUSE xxxxx message is printed, the operator should compare the number printed (i.e., xxxxx) with any operating instructions supplied by the programmer and per-

form the indicated operations. Upon completing the instructions, the operator should press the Interrupt Key to resume execution of the job.

INPUT/OUTPUT INTERRUPTION MESSAGES

There are three types of I/O interruption messages printed when an I/O device, except the 1052 Printer-Key-board, malfunctions:

1. FIS xxx.
2. FID xxx.
3. FIA xxx.

The xxx portion of the message represents the actual machine address of the device which caused the interruption.

If the 1052 Printer-Key-board is not functioning properly, printing is terminated.

Each message is displayed by the Instruction Counter (Storage Address) in addition to it being printed. The message is displayed in an internal representation of the external character set depending upon the machine conversion used (e.g., EBCDIC, BCDIC, ASCII, etc.).

FIS xxx

The FIS xxx message is displayed when any one of the following conditions occur:

1. Channel Data Check.
2. Channel Control Check.
3. Interface Control Check.
4. Equipment Check.

When the FIS xxx message is displayed, the operator should address the tape (or card reader) on which SEREP (System Environment Recording and Editing Program) was placed by using the Load Address Switch and press the Load Key. Once SEREP has completed its processing, the operator must re-initialize the FORTRAN system and perform the necessary actions to rerun the

error-interrupted FORTRAN job. (See the section, "Starting Instructions.")

FID xxx

The FID xxx message is displayed when any one of the following conditions occur:

1. Program Check.
2. Protection Check.
3. Command Reject.
4. Data Converted Check.

When the FID xxx message is displayed, the operator should press the Interrupt Key to retry the I/O device that caused the malfunction. If this is ineffective, the operator should terminate job processing.

FIA xxx

The FIA xxx message is displaced when any one of the following conditions occur:

1. Data Check.
2. Overrun by a tape device.
3. Bus Out Check.
4. Intervention required.
5. Chaining Check by a tape device.

When the FIA xxx message is displayed, the operator should make about five attempts to retry the I/O device that caused the malfunction by pressing the Interrupt Key. If this is ineffective, the operator should address the tape (or card reader) on which SEREP was placed by using the Load Address Switch and press the Load Key. Once SEREP has completed its processing, the operator must re-initialize the FORTRAN system and perform the necessary actions to rerun the error-interrupted FORTRAN job.

MACHINE CHECK MESSAGE

The hexadecimal characters C6D4E2, representing the characters FMS, are displayed in the low-order portion of the new Program Status Word (PSW) when a machine check is encountered. The entire PSW is displayed by the Instruction Counter (Storage Address) on the console.

When the characters C6D4E2 are displayed in the low-order portion of the PSW, the operator should address the tape (or card reader) on which SEREP was placed by using the Load Address Switch and press the Load Key. Once SEREP has completed its processing, the operator must re-initialize the FORTRAN system and perform the necessary actions to rerun the error-interrupted FORTRAN job. (See the section, "Starting Instructions.")

TYPES OF JOBS

There are four possible types of jobs that may be processed under control of the FORTRAN system:

1. Compile only.
2. Compile and execute.
3. Execute only (load).
4. Edit.

The type of job is specified on control cards accompanying the job. However, in the absence of control cards, the FORTRAN system assumes certain information about the job and, therefore, the operator need not add any control cards.

STARTING INSTRUCTIONS

The steps an operator should follow in preparing a machine for a FORTRAN job are:

1. Perform steps 1a, 1b, 1c, or 1d, depending on the type of job to be processed.
 - a. If edit, mount at least one work tape on the device designated by the programmer. Upon completion of the job, this tape is the new FORTRAN system tape. Additional tapes may have to be mounted on tape devices, depending on the number of new system tapes to be generated by the edit job. When the new system tape(s) are generated, the operator should remove the file protection ring from the tape(s).
 - b. If compile only, mount two work tapes: one on device 181, the other on 280. Go to step 2.

- c. If compile and execute, mount three work tapes: one on device 181, one on 280, and the third on 281. Device 281 (data set reference number 12) is the GO tape on which the object programs are placed. Go to step 2.
 - d. If execute only (i.e., load), place the GO tape on device 281 if the object programs are to be loaded from tape. Go to step 2.
2. Mount the FORTRAN system tape on any unused device that has the same number of tracks (7 or 9) as the one that

created the FORTRAN system tape. Go to step 3.

3. Using the Load Address Switch, address the FORTRAN system tape from the device upon which it was placed. Go to step 4.
4. Press Load Key.

The addresses of the actual tape devices upon which the operator places the FORTRAN system tape, work tapes, and GO tape may vary. In such cases, the operator should place the tapes on the devices specified by the programmer.

APPENDIX A: SOURCE PROGRAM DIAGNOSTICS

All diagnostic messages produced are printed in a group following the source program listing. Figure 34 shows the format of each message as it is printed under the heading of statement number, message number, and description.

When certain error conditions cannot be pinpointed to a single source statement, the error message contains an internal statement number of 0000.

Each message number (i.e., nnn) printed is preceded by the code 1F0031 as shown in Figure 34. For example, message number one is printed as:

S.xxxx 1F0031001 INVALID CONTROL CARD

The object program produced by the compilation is executed depending on:

1. The severity of the error in a particular source statement.
2. The option employed in the /JOB control statement, which determines the type of processing to be performed (i.e., GO or GOGO).

There are two types of diagnostic messages: (1) serious-error messages and (2) warning messages that an error may exist. If the GO option is specified and serious errors are encountered in the source program, execution of the object program is not attempted. If the GOGO option is specified and only warning messages are produced, the object program is executed. If the GOGO option is specified, object-program execution occurs irrespective of the type of errors existing in the source program.

A message whose number is immediately followed by the character "W" warns the user that an error may exist in the source statement. A serious error exists in the source statement if the message number is not followed by the character "W".

In the following text, each message number, the message itself, and its associated explanation is given. The abbreviated name preceding each explanation represents the segment of the FORTRAN system that generates the message.

The abbreviated name for each segment is:

FSD - FORTRAN System Director

CTL - Control Card Routine

P10 - Phase 10

P12 - Phase 12

P14 - Phase 14

P15 - Phase 15

P20 - Phase 20

IBC - IBCOM Routine

EDR - Editor

In some cases, more than one segment could generate the same message. For example, the message:

S.xxxx 1F0031001 INVALID CONTROL CARD

could be generated by either the editor or the control card routine; therefore, the explanation is preceded by the abbreviated names EDR and CTL.

<u>STATEMENT NUMBER</u>	<u>MESSAGE NUMBER</u>	<u>DESCRIPTION</u>
S.xxxx	1F0031nnn	message
xxxx	is the internal statement number	
nnn	is the message number	
message	is the actual message printed	

Figure 34. Format of Diagnostic Messages

001 INVALID CONTROL CARD

EDR and CTL -- Misspelling or punctuation error in the control statement makes card not recognizable.

002 CORE SIZE LESS THAN 16K - JOB ABORTED

EDR and CTL -- The FORTRAN system is unable to operate in this environment for a compilation and/or execution. The main storage size, denoted by x, should be in the range, $16K \leq x \leq 64K$.

003W NO /DATA CARD FOUND

CTL -- A /DATA statement should precede object program data cards. If there are no data cards, a /DATA statement should appear as the last statement of a compilation and/or execution job. Warning is given so that the data cards are not read in and considered part of the source or object program.

004 END OF DATA SET FOUND ON SYSTEM TAPE

CTL -- This is the result of a machine error. The job is terminated.

014 INVALID CHARACTER

EDR -- Invalid Character in the 12-2-9 REP control statement.

015 INVALID PHASE NAME

EDR -- An unknown segment name is specified on a 12-2-9 REP or 12-2-9 EDR control card. The job is terminated.

016 ILLEGAL END OF DATA SET

EDR -- An end of data set is sensed on the system input or system tape before editing was completed. The job is terminated.

017 xxxxxx NOT IN LIBRARY

EDR -- The name of the subprogram (i.e., xxxxxx) specified on a /AFTER or /DELETE statement was not found in the system library. Check for possible misspelling of subprogram name.

018 xxxxxx OUT OF SEQUENCE

EDR -- The library subprogram names (i.e., xxxxxx) specified on /AFTER or /DELETE statements must correspond to the sequence in which the subprograms exist in the library.

019 TOO MANY DELETE NAMES

EDR -- Not more than 21 subprograms may be deleted during a single edit job.

020 TOO MANY LIBRARY NAMES

EDR -- The number of library subprogram names permissible in the library depends on the main storage size of the machine. When main storage size is 16K, approximately 90 names are permitted.

029 ARRAY MUST BE DIMENSIONED ON ITS FIRST AND ONLY ITS FIRST OCCURRENCE

P10 -- The dimension of an array must be given prior to its use in any other statement and may never be re-defined. This can be done in either a DIMENSION statement, an explicit specification statement, or a COMMON statement.

030 ILLEGAL USE OF FUNCTION NAME

P10 -- A function name may not appear in an EQUIVALENCE or COMMON statement.

031 EQUIVALENCE OR COMMON TABLE FULL

P10 -- There are too many variables in COMMON and too many EQUIVALENCE relations specified. Approximately 60 variables may appear in COMMON and approximately 60 symbols may appear in EQUIVALENCE relations.

032 NUMBER TOO BIG

P10 -- Integer is larger than maximum size allowable (i.e., larger than $2^{31}-1$).

033W CARD MISSING

P10 -- First non-comment statement was a continuation line (i.e., a non-zero character, other than a blank, was encountered in column 6.) The statement is processed as if it were the initial line of a statement.

034 SUBPROGRAM CARD NOT FIRST

P10 -- A header card does not precede the subprogram statements to which it is associated. The first card in a subprogram (other than a comments card) is not a FUNCTION or SUBROUTINE statement.

035 ARGUMENT MISSING IN FUNCTION DEFINITION

P10 -- Function definition (either in an Arithmetic Statement Function or FUNCTION header statement) must have at least one argument.

036 ILLEGAL CHARACTER

P10 -- Delimiter is not recognizable. It should be either b + - * / =) , . (or Column 73, where b is a blank.

037 INVALID STATEMENT OR STATEMENT NUMBER

P10 -- An equal sign is missing in an Arithmetic Statement Function or an arithmetic statement is missing (e.g., in an IF Statement or an illegal delimiter precedes the statement).

038 SEQUENCE ERROR

P10 -- All declarative statements must precede all executable statements.

039 MORE THAN SIX CHARACTERS IN NAME

P10 -- Not allowed.

041 MULTI-DEFINED NAME

P10 -- A symbol is defined more than once. For example, a real variable is redefined as an integer variable.

042 MULTI-DEFINED STATEMENT NUMBER

P10 -- This statement number has been used previously. Every statement number should be unique, and associated with only one statement in a program.

043 RESERVED WORDS

P10 and P12 -- A reserved word may not be used as a variable, array, or subprogram name.

044 TOO MANY DECIMAL POINTS

P10 -- Not more than one decimal point may appear in a real or double-precision number.

045 DECIMAL POINT AFTER E

P10 -- A decimal point has been found in the E decimal exponent part of a real or double-precision number.

046 TOO MANY E'S

P10 -- A second E has been found in a number (e.g., 2.7E2E3).

047 ILLEGAL NUMBER OR NAME

P10 -- Illegal use of a number. For example, in the statement, DIMENSION 5(1,2), the number 5 is not a proper array name.

048 MORE THAN THREE DIMENSIONS

P10 -- Maximum number of dimensions permitted in an array is three.

049 DIMENSION ERROR

P10 -- Illegal delimiter or dimension of an array too big in a COMMON, DIMENSION, or Explicit Specification statement.

050 CANNOT EQUATE

P10 -- At least two variables or subscripted variables should appear in the parenthesis of an EQUIVALENCE statement.

051W COMMA MISSING

P10 -- A required comma was not encountered. The statement is compiled as though a comma were there.

052 WRONG DIMENSION

P10 -- Number of subscripts in the variable used does not correspond to the number of subscripts in the variable as defined in a COMMON, DIMENSION, or Explicit Specification statement.

053 SUBSCRIPT ERROR

P10 -- The subscript expression contains more than three subscripts, an illegal delimiter, or an illegal variable.

054 INVALID ARGUMENT IN ASF

P10 -- An illegal delimiter or variable appears in the statement function argument list.

055 INVALID ARGUMENT IN HEADER CARD

P10 -- An illegal variable or a multi-defined variable appears in the function definition argument list.

056 ILLEGAL STATEMENT NUMBER FIELD

P10 -- Statement number list in a computed GO TO or in an arithmetic IF statement is invalid.

057 DATA SET REFERENCE NUMBER MISSING

P10 -- There is no data set reference number specified. For example, WRITE (,10).

058 LEFT PARENTHESIS MISSING AFTER READ/WRITE

P10 -- The left parenthesis in a READ or WRITE statement is missing. For example, in the statement: WRITE 3,10), the left parenthesis before the 3 is needed.

060 ERROR IN VARIABLE

P10 -- Illegal variable in an EQUIVALENCE statement.

061W STATEMENT CANNOT BE REACHED

P10 -- Statement following a GO TO, RETURN, or STOP has no statement number.

063 EQUIVALENCE SUBSCRIPT ERROR

P10 -- There is an illegal delimiter or a missing subscript in an EQUIVALENCE subscript.

064 TOO MANY SYMBOLS AND STATEMENT NUMBERS

P10 -- The Dictionary Table has overflowed. Subdividing the program or reducing the number of symbols and statement numbers is necessary.

065 ILLEGAL STATEMENT NUMBER OR PAUSE NUMBER

P10 -- Either an alphabetic or illegal character is the first character in a statement number or there are more than five digits in the statement number.

066 BACKWARD DO LOOP

P10 -- The statement specified in the range of the DO statement may not precede the DO statement.

068 ERROR IN EXPONENT

P10 -- An exponent is missing or it is too big in a real or double-precision number.

069 TOO MANY ARGUMENTS IN ASF

P10 -- More than fifteen arguments in Arithmetic Statement Function definition is not permitted.

070 ILLEGAL FUNCTION NAME

P10 -- Illegal subprogram name in a FUNCTION or SUBROUTINE header statement.

071 ILLEGAL SUBROUTINE NAME

P10 -- Illegal delimiter or illegal subroutine name in a CALL statement.

072 ASF OUT OF SEQUENCE

P10 -- Arithmetic Statement Function statement is out of sequence or an array is not dimensioned prior to its first use.

073 TRANSFER TO NON-EXECUTABLE STATEMENT

P10 -- The statement number referred to by a GO TO, computed GO TO, or an arithmetic IF statement is a FORMAT or specification statement.

074 INCONSISTENT EQUATE

P10, P12 -- A variable appears in COMMON more than once or an inconsistent equate was made (e.g., the statement COMMON (A, B, C, A) is illegal).

075 UNFINISHED STATEMENT

P14 -- A right parenthesis at the end of the statement is missing.

076 PARENTHESIS ERROR

P10 and P14 -- A parenthesis is not closed or is missing. The error is substantial.

077 ILLEGAL DELIMITER OR MISSING SYMBOL

P10 and P14 -- An improper delimiter or illegal special character was encountered.

078 ILLEGAL END DO

P14 -- The last statement in the range of a DO loop cannot be a non-executable statement, Arithmetic IF, GO TO, PAUSE, RETURN, STOP, or another DO statement.

079 TYPE MUST BE INTEGER SCALAR

P10 and P14 -- The DO variable must be a non-subscripted integer variable.

080 COMMA MISSING

P14 -- A required comma was not encountered. The error is substantial.

081 ERROR IN PUNCTUATION

P10 and P14 -- Illegal decimal point or a number is missing following decimal point.

082 ILLEGAL NUMBER

P10 and P14 -- There is an error in an integer, real, or double precision number.

083 ERROR IN INTEGER

P10 and P14 -- Number zero not allowed.

084 MORE THAN FOUR WARNINGS IN STATEMENT

P10 and P14 -- Further attempt to compile the statement is terminated because four warning messages have already been generated.

085 COMPILER ERROR

P12 and P14 -- Compilers working text contains meaningless code. Job processing is continued.

086 ILLEGAL BLANK

P14 -- An illegal embedded blank was found in the FORMAT statement.

087 NUMBER MISSING

P14 -- A number is missing in E, F, T, A, I, D, or X conversion code or an illegal delimiter precedes the number.

088 NESTED PARENTHESIS

P14 -- Not more than one level of nested parentheses is permitted in a FORMAT statement.

089 ILLEGAL DATA SET REFERENCE NUMBER

P14 -- Data set reference number must be an integer variable or constant.

090 QUOTE NOT CLOSED

P14 -- A quote mark was not found terminating the literal data in a FORMAT statement.

091 ILLEGAL SIGN

P14 -- A P format code or a blank are the only legal delimiters following a plus or minus sign in a FORMAT statement.

092 ILLEGAL COMMA

P14 -- An erroneous comma appears in the statement.

093 NUMBER TOO BIG

P14 -- The number specified in the FORMAT statement either preceding or following the format code cannot be longer than four digits.

094 TOO MANY DECIMAL PLACES

P14 -- The number of decimal places must be less than the size of the entire number in a FORMAT statement.

095 STATEMENT NUMBER REFERENCE NOT FORMAT STATEMENT

P14 -- The statement number referred to in a READ/WRITE statement is not that of a FORMAT statement.

096 ILLEGAL VARIABLE IN I/O LIST

P14 -- The use of subprogram names is not allowed in an I/O list.

097 TOO MANY ELEMENTS IN I/O LIST

P14 -- The I/O list in the READ or WRITE statement contains too many elements. There are approximately 250 variables permitted in a single I/O list. The READ or WRITE statement should be divided into several statements.

098 NO CHARACTER BETWEEN QUOTES

P14 -- An open quote is immediately followed by close quote in the FORMAT statement. At least one character should appear within the quotes.

099 TOO MANY CHARACTERS BETWEEN QUOTES

P14 -- The number of characters appearing within quotes in the FORMAT statement is too large for the compiler to handle. That is, not more than 255 characters should appear between quote marks in a FORMAT statement. The format quote usage should be subdivided.

100 ILLEGAL DO VARIABLE OR CONSTANT

P14 -- DO parameter must be an integer non-subscripted variable or integer constant.

123 FUNCTION NAME NOT DEFINED

P15 -- The function name is not defined in its function subprogram (i.e., it does not appear on the left side of an equal sign).

124 ERROR IN FUNCTION CALL

P15 -- An undetermined error was found in the argument list of a function call.

125 DO VARIABLE REDEFINED

P15 -- A DO variable has been redefined within the range of the DO loop.

126 FUNCTION ARGUMENT MISSING

P15 -- An argument of a function reference is missing.

127 COMPILER ERROR

P12 and P15 -- Compilers working text contains meaningless code. Job processing is terminated.

128 INVALID CALL OR IF STATEMENT

P15 -- Illegal call, or Arithmetic IF statement.

129 MULTI-DEFINED NAME

P15 -- A name is re-defined.

130 INVALID ARGUMENT

P15 -- The mode of the argument does not agree with the mode of the function.

131 WRONG MODE

P15 -- The mode of the argument does not agree with the mode of the in-line function.

132 INCORRECT NESTING OF DO

P15 -- The last statement in the range of the DO loop nested within other DO loops, exceeds the range of one or more of those DO loops.

133 ILLEGAL EQUAL SIGN

P15 -- Two equal signs appear in the same statement.

134 MORE THAN FOUR WARNINGS IN STATEMENT

P15 -- Further attempt to compile the statement is terminated because four warning messages have already been generated.

135 SUBSCRIPT NOT ALLOWED

P15 -- A subscript is not allowed in an arithmetic statement function definition.

136 UNDEFINED STATEMENT NUMBER

P15 -- The referenced statement number does not exist in the program.

137 NAME MISSING OR ILLEGAL DELIMITER

P15 -- Illegal delimiter found. For example, $X=A*B$. A variable or constant is missing between the two operators or one of the operators is superfluous.

138 SUPERFLUOUS OPERATOR

P15 -- There is a superfluous arithmetic operator in the statement.

139 WRONG NUMBER OF ARGUMENTS IN ASF CALL

P15 -- The number of arguments in an Arithmetic Statement Function reference or in an in-line function reference does not agree with the function definition.

140 TOO MANY PARAMETERS

P15 -- The maximum number of parameters allowed is 48. Subdividing the statement is suggested.

141 ILLEGAL SUBPROGRAM NAME

P15 -- Name of a function or subroutine call is not defined as a function or subroutine subprogram name.

142 MORE THAN TWENTY FIVE LEVELS OF DO NESTING

P15 -- Not more than twenty-five DO loops may be nested.

143 INVALID RESULT FIELD

P15 -- The result field of an arithmetic statement is invalid.

144 ILLEGAL NUMBER OF STATEMENT NUMBERS

P15 -- An arithmetic IF statement must contain exactly three statement numbers.

145 PROGRAM TOO BIG

P12 and P20 -- The object program has exceeded the basic register range.

146 INCONSISTENT EQUATE

P12 -- For example, EQUIVALENCE (A(1),B), (A(2),C), (B,C) or a double precision variable in COMMON is not on boundary.

147 TWO VARIABLES IN COMMON

P12 -- Two or more variables equivalenced are in common.

148 COMMON EXTENDED UPWARD

P12 -- An EQUIVALENCE statement causes COMMON to be extended in negative direction.

149 DUMMY ARRAY OR VARIABLE IN COMMON

P12 -- A dummy variable or array is not permitted in COMMON.

0150 CANNOT EQUATE

P12 -- The equated name is not a variable.

160W SUPERFLUOUS INFORMATION AT END OF STATEMENT

P10, P14 and P15 -- The statement has been completed but something superfluous exists at the end (e.g., Rewind I XYZ, XYZ is superfluous).

161W SUGGEST SUBDIVIDING PROGRAM

P12 -- Program causes use of spill base register producing inefficient object code.

162W BLANK CARD

P10 -- The card contains only a statement number. The card is ignored.

163W TOO MANY DIGITS IN NUMBER

P10 -- There are too many significant decimal digits in number.

164W STATEMENT NUMBER MISSING

P10 and P14 -- Format statement must have a statement number. The statement is ignored.

165W UNREFERENCED FORMAT STATEMENT

P14 -- A FORMAT statement is not referred to by any other statement. The FORMAT statement is not processed.

166W REDUNDANT COMMA

P10 and P14 -- The redundant comma in the statement is ignored.

167W LINE TOO LONG

P14 -- Format line length exceeds printer line length. Processing is completed.

168W END CARD MISSING

P10 -- The END statement is missing. Job processing continues as if it were there.

169W RIGHT PARENTHESIS MISSING

P10 and P15 -- The right parenthesis in the statement is missing. Processing continues as if it were there.

170W ZERO OR NO COUNT IN X CONVERSION

P14 -- The number preceding the X format code is 0 or blank. Processing continues.

171W PARAMETERS MISSING

P10 and P14 -- There are no parameters following a left parenthesis or a comma.

172W UNREFERENCED ASF ARGUMENT

P10 -- Argument or statement function not referred to in the arithmetic expression of an arithmetic statement function.

173W EXCESSIVE RIGHT PARENTHESIS

P10 -- The additional right parentheses in the statement are ignored and processing continues.

174W ARRAY USED AS SCALAR

P15 -- The name of the array is not followed by a subscript enclosed in parentheses.

175W STATEMENT NUMBER ON DECLARATIVE STATEMENT

P10 -- The statement number associated with the declarative statement is superfluous.

APPENDIX B: OBJECT PROGRAM DIAGNOSTICS

ERROR CODE DIAGNOSTICS

When an error condition arises during execution of a program, an error code number is printed. In the following text, the error codes are given in numerical order; each is followed by an explanation describing the type of error. Preceding each explanation is an abbreviated name indicating the segment of the FORTRAN system that generates the error code.

The abbreviated name for each segment is:

IBC -- IBCOM Routine

LDR -- Relocating Loader

LIB -- FORTRAN Library

200

LDR -- An end-of-data-set has been detected.

201

LDR -- An error has occurred in the IBCOM or reference table overlay process.

202

LDR -- An SLC, ICS, or REP statement has an invalid format.

203

LDR -- An SLC name was omitted from the SLC statement.

204

LDR -- A program to which reference was made by another program is missing.

205

LDR -- An error has occurred in the blank COMMON overlay process.

206

LDR -- An invalid ESD statement type has been detected.

207

LDR -- A duplicate symbol has been detected.

208

LDR -- The size of main storage is insufficient to process the job.

209

LDR -- An invalid loader card has been detected.

211

IBC -- An invalid character has been detected in a FORMAT statement.

212

IBC -- An attempt has been made to READ or WRITE past an end-of-data-set.

213

IBC -- The input list in a non-formatted I/O statement is larger than the logical record.

214

IBC -- An attempt has been made to WRITE more than 255 records within one logical record.

215

IBC -- An invalid character exists for the decimal input corresponding to an I,E,F or D format code.

216

IBC -- An illegal sense light number was detected in a SLITE or SLITET statement.

217

IBC -- An end-of-data-set was sensed during a READ operation.

241

LIB -- In the subprogram FIXPI(I**J), where I and J are integer variables or subscripted variables, I=0, J≤0 is illegal.

242

LIB -- In the subprogram FRXPI(R**J), where R and J are real and integer variables or subscripted variables, respectively, R=0, J≤0 is illegal.

243

LIB -- In the subprogram FDXPI(D**J), where D and J are double-precision and integer variables or subscripted variables, respectively, D=0, J≤0 illegal.

244

LIB -- In the subprogram FRXPR(R**S), where R and S are real variables or subscripted variables, R=0, S≤0 is illegal.

245

LIB -- In the subprogram FDXPD(D**P), where D and P are double-precision variables or subscripted variables, D=0, P≤0 is illegal.

251

LIB -- In the subprogram SQRT(x), x<0 is illegal.

252

LIB -- In the subprogram EXP(x), x>174.673 is illegal.

253

LIB -- In the subprogram LOG(x) or LOG10(x), x≤0 is illegal.

254

LIB -- In the subprogram SIN(x) or COS(x), |x| ≥ 2¹⁸ is illegal.

261

LIB -- In the subprogram DSQRT(x), x<0 is illegal.

262

LIB -- In the subprogram DEXP(x), x>174.673 is illegal.

263

LIB -- In the subprogram DLOG(x) or DLOG10(x), x≤0 is illegal.

264

LIB -- In the subprogram DSIN(x) or DCOS(x), |x| ≥ 2⁵⁰ is illegal.

PROGRAM INTERRUPT MESSAGES

Program interrupt messages containing the old Program Status Word (PSW) are produced when one of the following occurs:

1. Exponent-Overflow Exception.
2. Exponent-Underflow Exception.
3. Floating-Point-Divide Exception.

Operator intervention is not required for any of these interruptions. Figure 35 shows the interruption message format.

The three characters in the PSW (i.e., C, D, or F) represents the code number (in hexadecimal) associated with the type of interruption.

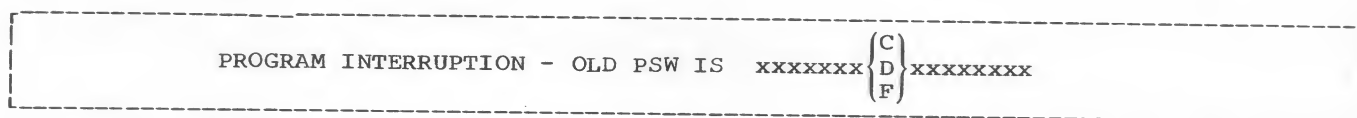


Figure 35. Program Interrupt Message Format

Exponent-Overflow Exception

The exponent-overflow exception, assigned code number C, is recognized when the result of a floating-point addition, subtraction, multiplication, or division is greater than 16^{63} (approximately 10^{75}).

Exponent-Underflow Exception

The exponent-underflow exception, assigned code number D, is recognized when

the result of a floating-point addition, subtraction, multiplication, or division is less than 16^{-63} (approximately 10^{-75}).

Floating-Point-Divide Exception

The floating-point-divide exception, assigned code number F, is recognized when division by a floating-point number with a zero fraction is attempted. For more information about the PSW, refer to the publication, IBM System/360 Principles of Operation, Form A22-6821.

This appendix provides information needed to prepare and use relocatable subprograms written in assembler language with a FORTRAN job.

CALLED AND CALLING PROGRAMS

Any subprogram that is referred to by another program is considered a called program; if this called subprogram refers to another subprogram then it is both a called and calling subprogram. In Figure 36, for example, if program A calls program B and program B calls program C then:

1. A is considered a calling program by B.
2. B is considered a called program by A.
3. B is considered a calling program by C.
4. C is considered a called program by B.

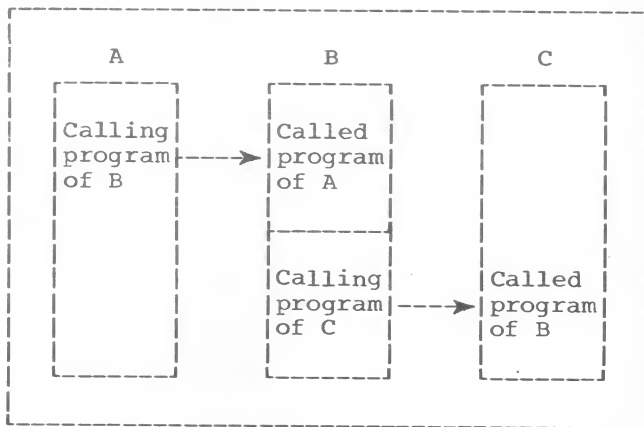


Figure 36. Called and Calling Programs

There are three basic FORTRAN job structures that can be formed using assembler-written subprograms in a FORTRAN job:

1. A FORTRAN program (or subprogram) calling an assembler-written subprogram.
2. An assembler-written subprogram calling a FORTRAN subprogram.

3. An assembler-written subprogram calling another assembler-written subprogram.

From these combinations, more complicated structures may be formed. For example, a FORTRAN program can call an assembler-written subprogram which then could call another assembler-written subprogram.

The Basic Programming Support FORTRAN System has established certain conventions which must be considered when giving and returning control to assembler-written subprograms. These conventions, called linkage conventions, are described in the following text.

FORTRAN LINKAGE CONVENTIONS

When a FORTRAN subprogram calls another FORTRAN subprogram, certain save and return routines are generated in addition to a calling sequence that actually transfers control. Figure 37 shows a typical subprogram configuration generated by a calling subprogram.

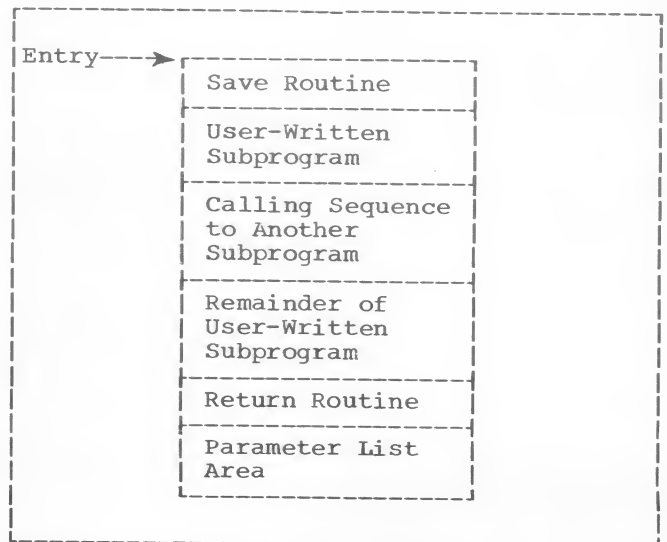


Figure 37. Calling Subprogram Configuration

Assembler-written subprograms need not be constructed with save and return routines exactly as the FORTRAN system generates them; however, there are basic conven-

tions of the FORTRAN system to which the assembler programmer must adhere. These conventions include:

1. Utilizing the proper registers in establishing a linkage.
2. Reserving a parameter area in the calling program in which the called program may refer to the parameter list.
3. Reserving a save area in which the registers used in the linkage may be saved.

REGISTER USE

The Basic Programming Support FORTRAN System has assigned roles to certain registers used in linkages. Figure 38 illustrates the function of each linkage register.

PARAMETER AREA

Every assembled subprogram which calls another subprogram must reserve an area of storage (parameter area) in which the parameter list used by the called subprogram

is located. Each entry in the parameter area occupies four bytes at a full-word boundary.

The first byte (bit 0 through bit 7) of each entry in the parameter area contains zeros. The bit 0, however, may contain a 1 to indicate the last entry.

The last three bytes of each entry contain the 24 bit address of the argument.

SAVE AREA

An assembled subprogram, which calls another subprogram, must reserve an area of storage (save area) in which certain registers (i.e., those used in the subprogram and those used in the linkage to the subprogram) are saved.

The maximum amount of storage reserved by the subprogram is 18 words. Figure 39 shows the layout of the save area and the contents of each word.

An assembled subprogram which does not call another subprogram need not establish a save area. However, if registers 13 or 14 are used by the subprogram, that subprogram should save their contents in a desired location and restore them before returning control to the calling program.

Register Number	Register Name	Contents
0	Result Register	Used for function subprograms only. The result is returned in general or floating-point register 0, depending on the type of function (e.g., integer, real, and double-precision). For subroutine subprograms, the result(s) is returned in the variables specified by the programmer.
1	Parameter List Register	Address of the parameter list passed to the called program.
13	Save Area Register	Address of the area reserved by the program being executed (called or calling) in which the contents of certain registers are stored.
14	Return Register	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry Point Register	Address of the entry point in the called program.

Figure 38. Linkage Registers

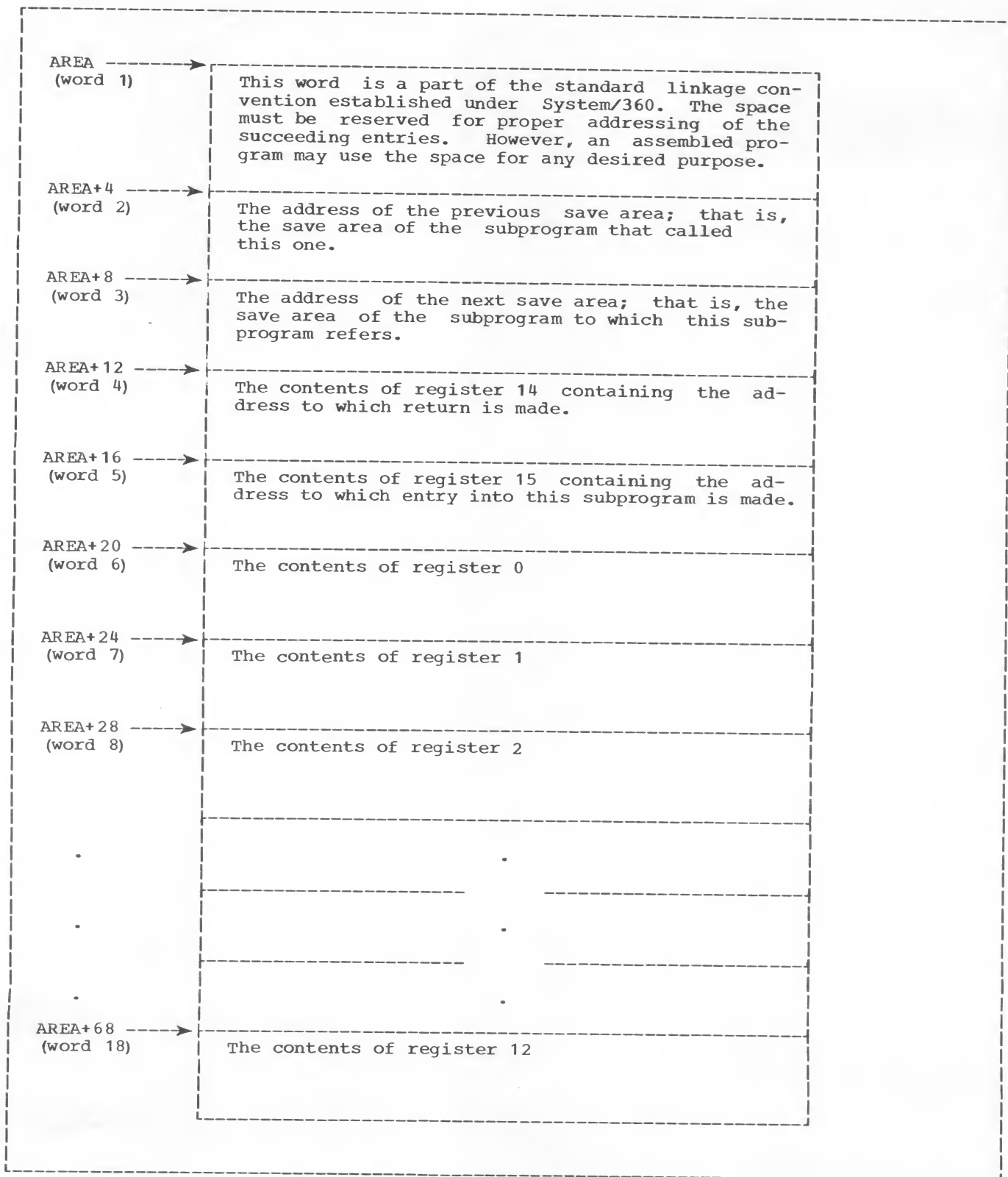


Figure 39. Save Area Layout and Word Contents

SAMPLE CALLING SUBPROGRAM LINKAGE

The linkage conventions used by an assembled subprogram which calls another subprogram are shown in Figure 40.

The coding does not have to exactly conform to that shown in Figure 40. However, the linkage should include:

1. The calling sequence by which an assembled subprogram may reference another subprogram.

2. The save and return routines by which the appropriate save area is established and control is returned to the calling program.

3. The out-of-line parameter area by which an assembled subprogram may pass parameters. (An in-line parameter area may be used instead; see the section, "In-line Parameter Area.")

deckname	START	0	
	ENTRY	name ₁	
	EXTRN	name ₂	
	DC	CLm'name ₁ '	m must be an odd number to ensure that the program starts on a half-word boundary. The name may be padded with blanks.
	DC	X'm'	
*			
	USING	*,15	
*	Save Routine		
name ₁	STM	14,r ₁ ,12(13)	the contents of registers 14, 15, and 0 through r ₁ are stored in the save area of the calling program (previous save area). r ₁ is any number from 0 through 12.
*			
*			
*	LR	r ₂ ,13	loads register 13, which points to the save area of the calling program, into any general register, r ₂ , except 0 and 13.
*			
*	LA	13,AREA	loads the address of this program's save area into register 13.
*			
*	ST	13,8(0,r ₂)	store the address of this program's save area into word 3 of the save area of the calling program.
*			
*	ST	r ₂ ,4(0,13)	stores the address of the previous save area (i.e., the same area of the calling program) into word 2 of this program's save area.
*			
AREA	BC	15,prob ₁	
*	DS	18F	reserves 18 words for the save area. This is last statement of save routine.
prob ₁	User-written program statements		
	.		
	.		
*	Calling Sequence		
	LA	1,ARGLST	first statement in calling sequence.
	L	15,ADCON	
	BALR	14,15	
*	BC	0,X'isn'	last statement in calling sequence. The isn represents the internal statement number that is used by a trace routine.
*			
	.		
	.		
*	Remainder of user-written program statements		
	.		
	.		
*	Return Routine		
	L	13,AREA+4	first statement in return routine. Loads the address of the previous save area back into register 13.
*			
*	LM	2,r ₁ ,28(13)	the contents of registers 2 through r ₁ are restored from the previous save area.
*			
*	L	14,12(13)	loads the return address, which is in word 4 of the calling program, into register 14.
*	MVI	12(13),X'FF'	sets flag FF in the save area of the calling program to indicate that control has returned to the calling program.
*			
	BCR	15,14	last statement in return routine.
ADCON	DC	A(name ₂)	contains the address of subprogram name ₂ .
*	Parameter Area		
ARGLST	DC	AL4(arg ₁)	first statement in parameter area setup.
	DC	AL4(arg ₂)	
		.	
		.	
		.	
	DC	X'80'	first byte of last argument.
	DC	AL3(arg _n)	last statement in parameter area setup.

Figure 40. Sample Linkage Routines Used With a Calling Subprogram

Lowest Level Subprograms

If an assembled subprogram does not call any other program (i.e., if it is at the lowest level), the programmer should omit the save routine, calling sequence and parameter area shown in Figure 40. Figure 41 shows the appropriate linkage conventions used by an assembled subprogram at the lowest level.

```

deckname      START      0
              ENTRY      name
              DC          CLm'name'
              DC          X'm'
              USING      *,15
              STM        14,r1,12(13)
              .
              .
              .
User - written program statements
              .
              .
              .
              LM          2,r1,28(13)
              MVI        12(13),X'FF'
              BCR        15,14

```

Note: If registers 13 and/or 14 are used in the called subprogram, their contents should be saved and restored by the called subprogram.

Figure 41. Sample Linkage Routines Used
With a Lowest Level Subprogram

In-Line Parameter Area

The assembler programmer may establish an in-line parameter area instead of out-of-line area. In this case, he may

substitute the calling sequence and parameter area shown in Figure 40 with that shown in Figure 42.

ADCON	DC	A (prob ₁)
	.	
	.	
	.	
	LA	14, RETURN
	L	15, ADCON
	CNOP	2, 4
	BALR	1, 15
	DC	AL4 (arg ₁)
	DC	AL4 (arg ₂)
	.	
	.	
	.	
	DC	X'80'
	DC	AL3 (arg _n)
RETURN	BC	0, X'isn'

Figure 42. Sample In-Line Parameter Area

SHARING DATA IN COMMON

The Basic Programming Support FORTRAN System uses register 4 as a pointer to a COMMON area of a FORTRAN program. Additional registers (e.g., register 5, 6, etc.) are used if the size of the COMMON area warrants such.

If an assembler-written subprogram is to share data in COMMON with a FORTRAN program, that subprogram should also use register 4. However, if an assembler-written subprogram is to share data in COMMON with another assembler-written subprogram, the programmer may use any general register available as a pointer to the COMMON area.

There are two types of subprograms provided in the IBM System/360 Basic Programming Support FORTRAN IV library: mathematical subprograms and service (utility) subprograms. This section contains information to help the programmer select the proper subprogram for solving his problem.

DEFINITION OF SYMBOLS

The symbols used throughout this appendix are:

Symbol	Explanation
$g(x)$	= The result given by the subprogram.
$f(x)$	= The correct extra precision result.

The symbols used in describing the effect of an argument error on the accuracy of the result given by the subprogram are:

Symbol	Explanation
ϵ	= $\left \frac{f(x) - g(x)}{f(x)} \right $ The relative error of the result given by the subprogram.
δ	= The relative error of the argument.
E	= $ f(x) - g(x) $ The absolute error of the result given by the subprogram.
Δ	= The absolute error of the argument.

The symbols used in describing the accuracy of the result given by the subprogram are:

Symbol	Explanation
$M(E)$	= $\text{Max} f(x) - g(x) $ The maximum absolute error produced during testing.
$M(\epsilon)$	= $\text{Max} \left \frac{f(x) - g(x)}{f(x)} \right $

The maximum relative error produced during testing.

$$\sigma(E) = \sqrt{\frac{1}{N} \sum_i |f(x_i) - g(x_i)|^2}$$

The root-mean-square (standard deviation) absolute error.

$$\sigma(\epsilon) = \sqrt{\frac{1}{N} \sum_i \left| \frac{f(x_i) - g(x_i)}{f(x_i)} \right|^2}$$

The root-mean-square (standard deviation) relative error.

MATHEMATICAL SUBPROGRAM DESCRIPTIONS

To facilitate quick reference, the following description of the mathematical subprograms are arranged in the order in which they appear on the FORTRAN system tape supplied by IBM. The description of each mathematical subprogram includes the:

1. Purpose.
2. Permissible Entry Points.
3. Range.
4. Accuracy.
5. Storage requirements.
6. Considerations that should be noted in using the subprogram.
7. Method by which each subprogram is derived.
8. Calling Sequence.

ACCURACY

Because the size of a machine word is limited, small errors may be generated by mathematical subroutines. In an elaborate computation, slight inaccuracies can accumulate and become large errors. Thus, in interpreting final results, the user should take into account any errors introduced during the various intermediate stages.

The accuracy of an answer produced by a subroutine is influenced by two factors: (1) the accuracy of the argument, and (2) the performance of the subroutine.

The Accuracy of the Argument

Most arguments contain errors. An error in a given argument may have accumulated over several steps prior to the use of the subroutine. Even data fresh from input conversion contain slight errors. Because decimal data cannot usually be exactly converted into the binary form required by the processing unit the conversion process is usually only approximate. Argument errors always influence the accuracy of answers. The effect of an argument error on the accuracy of an answer depends solely on the nature of the mathematical function involved and not on the particular coding by which that function is computed within a subroutine. In order to assist users in assessing the accumulation of errors, a guide on the propagational effect of argument errors is provided for each function. Wherever possible, this is expressed as a simple formula.

The Performance of the Subroutine

The performance statistics supplied in this appendix are based upon the assumption that arguments are perfect (i.e., without errors, and therefore having no argument error propagation effect upon answers). Thus the only errors in answers are those introduced by the subroutines themselves.

For each subroutine, accuracy figures are given for one or more representative segments within the valid argument range(s). In each case the particular

statistics given are those most meaningful to the function and range under consideration.

For example, the maximum relative error and standard deviation of the relative error of a set of answers are generally useful and revealing statistics, but useless for the range of a function where its value becomes 0, since the slightest error of the argument value can cause an unbounded fluctuation on the relative magnitude of the answer. Such is the case with $\sin(x)$ for x near Π , and in this range it is more appropriate to discuss absolute errors.

METHOD

Some of the formulas are widely known; others not so widely known are derived from more commonly known formulas. In such cases, the steps leading from the common formula to the computational formula have been sketched with enough detail so that the derivation may be reconstructed by anyone with a basic understanding of mathematics and who has access to the common texts on numerical analysis.

Any of the common numerical analysis texts may be used as a reference. One such text is Hildebrand, F.B., Introduction to Numerical Analysis, McGraw-Hill Book Company Inc., New York, N. Y., 1956. Background information for algorithms involving continued fractions may be found in Wall, H.S., Analytic Theory of Continued Fractions, D. Van Nostrand Co., Inc., Princeton, N. J., 1948.

ALOG Subprogram

PURPOSE: To compute the natural (ALOG) or the common (ALOG10) logarithm of a real number.

ENTRY POINTS: ALOG and ALOG10

RANGE: $0 < x$

ACCURACY: The accuracy of the ALOG subprogram is shown in Figures 43 and 44.

ARGUMENT RANGE	$\sigma(E)$	$M(E)$
	ROOT-MEAN-SQUARE Absolute ERROR	MAXIMUM Absolute ERROR
ALOG		
$0.5 \leq x \leq 1.5$	8.62×10^{-8}	3.46×10^{-7}
ALOG10		
$0.5 \leq x \leq 1.5$	4.78×10^{-8}	1.64×10^{-7}
These statistics are based on a uniformly distributed argument sample.		

Figure 43. ALOG Subprogram, Absolute Error

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
ALOG		
x outside (0.5, 1.5)	1.20×10^{-7}	8.32×10^{-7}
ALOG10		
x outside (0.5, 1.5)	2.17×10^{-7}	1.05×10^{-6}
These statistics are based on an exponentially distributed argument sample.		

Figure 44. ALOG Subprogram, Relative Error

STORAGE REQUIREMENTS: 257 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. Write $x = 16^p \cdot m$, $\frac{1}{16} \leq m < 1$.

2. Define 2 constants a, b (a =base point, $2^{-b}=a$) as follows:

$$\text{If } \frac{1}{16} \leq m < \frac{1}{8}, \quad a = \frac{1}{16} \text{ and } b = 4$$

$$\text{If } \frac{1}{8} \leq m < \frac{1}{2}, \quad a = \frac{1}{4} \text{ and } b = 2$$

$$\text{If } \frac{1}{2} \leq m < 1, \quad a = 1 \text{ and } b = 0.$$

Write $z = \frac{m-a}{m+a}$. Then $m = a \cdot \frac{1+z}{1-z}$, and $|z| \leq \frac{1}{3}$.

3. Now $x = 2^{4p-b} \cdot \left(\frac{1+z}{1-z}\right)$. Hence $\log_e x = (4p-b) \log_e 2 + \log_e \left(\frac{1+z}{1-z}\right)$.

4. $\log_e \left(\frac{1+z}{1-z}\right)$ is computed using the Chebyshev interpolation polynomial of degree 4 in z^2 for the range $0 \leq z^2 \leq \frac{1}{9}$.
5. The maximum relative error of this approximation is $2^{-27.8}$.

5. $\log_{10} x = \log_{10} e \cdot \log_e x$.

The effect of an argument error is $E \sim \delta$. In particular, if δ is the minimal round-off error of the argument, say $6 \cdot 10^{-8}$, then $E \sim 6 \cdot 10^{-8}$. This means that if the argument is close to 1, the relative error can be very large, since the function value there is very small.

CALLING SEQUENCE: Out-of-line.

SQRT Subprogram

PURPOSE: To compute the square root of a real number.

ENTRY POINT: SQRT

RANGE: $0 \leq x$

ACCURACY: The accuracy of the SQRT subprogram is shown in Figure 45.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
The full range	1.68×10^{-7}	8.70×10^{-7}
These statistics are based on an exponentially distributed argument sample.		

Figure 45. SQRT Subprogram, Relative Error

STORAGE REQUIREMENTS: 168 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. If $x=0$, $\sqrt{x} = 0$. Otherwise write $x = 16^{2p \cdot m}$ where p is an integer and $\frac{1}{256} \leq m < 1$. Then $\sqrt{x} = 16^p \cdot \sqrt{m}$ and p and \sqrt{m} are the exponent and the mantissa of the answer respectively.

2. For the 1st approximation of \sqrt{m} , take one of the following two hyperbolic approximations of the form $a + b/(c + m)$:

a. For $\frac{1}{16} \leq m < 1$, the values $a = 1.80713$, $b = -1.57727$, $c = 0.954182$ minimize the maximal relative error ϵ_0 over the range while making the exact fit at $m=1$. The exact fitting at $m=1$ minimizes the computational loss of the last hexadecimal digit for the values of m slightly less than 1. $\epsilon_0 < 2^{-5.44}$.

b. For $\frac{1}{256} \leq m < \frac{1}{16}$, the values $a = 0.428795$, $b = -0.0214398$, $c = 0.0548470$ minimize $m^{\frac{1}{2}} \cdot \epsilon_0$ over the range where ϵ_0 denotes the relative error. $\epsilon_0 < 2^{-6.5} \cdot m^{-\frac{1}{2}}$.

3. Apply Newtown-Raphson iteration, $y_{n+1} = \frac{1}{2}(y_n + \frac{x}{y_n})$ twice to the 1st approximation y_0 . For $\frac{1}{16} \leq m < 1$, the final relative error is theoretically less than $2^{-24.7}$. For $\frac{1}{256} \leq m < \frac{1}{16}$, the final absolute error is theoretically less than 2^{-29} .

The effect of an argument error is $\sim \frac{1}{2}\delta$.

CALLING SEQUENCE: Out-of-line.

ATAN Subprogram

PURPOSE: To compute the principal value (in radians) of the arctangent of a real number.

ENTRY POINT: ATAN

RANGE: Any size real argument is acceptable to this subprogram.

ACCURACY: The accuracy of the ATAN subprogram is shown in Figure 46.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
The full range	4.54×10^{-7}	9.75×10^{-7}
These statistics are based on tangents of uniformly distributed numbers between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.		

Figure 46. ATAN Subprogram, Relative Error

STORAGE REQUIREMENTS: 192 bytes.

METHOD:

1. Reduce computation of $\text{ATAN}(x)$ to the case $0 \leq x \leq 1$ by using $\text{Atan}(-x) =$

$$-\text{Atan}(x), \text{Atan}\left(\frac{1}{|x|}\right) = \frac{\pi}{2} - \text{Atan}|x|.$$

2. Reduce further to the case $|x| \leq \tan 15^\circ = 0.26795$ by $\text{Atan}(x) = 30^\circ + \text{Atan}$

$$\frac{\sqrt{3}x-1}{x+\sqrt{3}}, \left| \frac{\sqrt{3}x-1}{x+\sqrt{3}} \right| \leq \tan 15^\circ \text{ if } \tan 15^\circ < x \leq 1.$$

Here compute $\sqrt{3}x-1$ as $(\sqrt{3}-1)x-1+x$ to avoid the loss of significant digits.

3. For $|x| \leq \tan 15^\circ$, use the approximation formula:

$$\frac{\text{Atan}(x)}{x} \cong 0.60310579 - 0.05160454x^2 + \frac{0.55913709}{x^2+1.4087812} \quad (*)$$

This formula can be obtained by transforming the continued fraction:

$$\frac{\text{Atan}(x)}{x} = 1 - \frac{1}{3}x^2 + \frac{\frac{1}{5}x^2}{\frac{7}{5} + x^2 - w}, \quad \text{after}$$

$$\text{substituting } \left(\frac{-75}{77} - x^{-2} + \frac{3375}{77} \right) \cdot 10^{-4} \text{ for } w = \frac{4 \cdot 5}{7 \cdot 7 \cdot 9} \left/ \left(\frac{43}{7 \cdot 11} + x^{-2} + \dots \right) \right.$$

The original continued fraction can be obtained by transforming the Taylor series into a continued fraction form. The relative error of the formula (*) is less than $2^{-27} \cdot 1$.

The effect of an argument error is $E \sim \Delta / (1+x^2)$. For small $x, \epsilon \sim \delta$; and as x becomes large, the effect of δ on ϵ diminishes.

CALLING SEQUENCE: Out-of-line.

TANH Subprogram

PURPOSE: To compute the hyperbolic tangent of a real number.

ENTRY POINT: TANH

RANGE: Any size real argument is acceptable to this subprogram.

ACCURACY: The accuracy of the TANH subprogram is shown in Figure 47.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
$ x \leq 0.54931$	1.66×10^{-7}	8.12×10^{-7}
$0.54931 < x \leq 5$	7.53×10^{-8}	5.74×10^{-7}
These statistics are based on a uniformly distributed argument sample.		

Figure 47. TANH Subprogram, Relative Error

STORAGE REQUIREMENTS: 260 bytes.

CONSIDERATIONS: The subprogram EXP is used by this subprogram.

METHOD:

- For $|x| \leq 2^{-12}$, give $\tanh x \cong x$.
- For $2^{-12} < |x| < 0.54931$, use the following fractional approximation:

$$\frac{\tanh x}{x} = 1 - \frac{x^2 + 35.1535}{x^2 + 45.1842 + \frac{105.4605}{x^2}}$$

This formula can be obtained by transforming the continued fraction:

$$\frac{1}{1 + \frac{x^2}{3} + \frac{x^2}{5} + \frac{x^2}{7} + w}$$

with an approximate value 0.0307 for w .

The relative error of this approximation is less than 2^{-27} .

- For $0.54931 \leq x < 9.011$, use $\tanh x = 1 - 2/(e^{2x} + 1)$.
- For $9.011 \leq x$, use $\tanh x \cong 1$.
- For $x \leq -0.54931$, use $\tanh x = -\tanh(-x)$.
- The exponential subroutine is used in the case 3 above.

The effect of an argument error is $E \sim (1 - \tanh^2 x) \Delta, \epsilon \sim 2 \Delta / \sinh 2x$. Thus, for small $x, \epsilon \sim \delta$, and as x gets larger, the effect of δ on ϵ diminishes.

CALLING SEQUENCE: Out-of-line.

EXP Subprogram

PURPOSE: To compute the value of "e" raised to the power of a real argument.

ENTRY POINT: EXP

RANGE: $x < 174.673$

ACCURACY: The accuracy of the EXP subprogram is shown in Figure 48.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
$ x \leq 1$	1.28×10^{-7}	4.65×10^{-7}
$ x \leq 170$	1.17×10^{-7}	4.69×10^{-7}
These statistics are based on a uniformly distributed argument sample.		

Figure 48. EXP Subprogram, Relative Error

STORAGE REQUIREMENTS: 280 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD

1. If $x < -180.218$, give 0 as the answer.
2. If $|x| < 2^{-28}$, give 1 as the answer.
3. Otherwise, divide x by $\log_e 2$ and write $y = x/\log_e 2 = 4a-b-d$, where a, b are integers, $0 \leq b \leq 3$, $0 \leq d < 1$. Then $e^x = 2^y = 16^a \cdot 2^{-b} \cdot 2^{-d}$.
4. Compute 2^{-d} by the following fractional approximation:

$$2^{-d} \approx$$

$$1 - \frac{2d}{0.034657359d^2 + d + 9.9545958 - 617.97227d + 87.417497}$$

This formula can be obtained by transforming the well known continued fraction:

$$e \approx \frac{1}{1} - \frac{z}{1} + \frac{z}{2} - \frac{z}{3} + \frac{z}{2} - \frac{z}{5} +$$

$$\frac{z}{2} - \frac{z}{7} + \frac{z}{2}$$

The maximum relative error of this approximation is 2^{-29} .

5. Multiply 2^{-d} by 2^{-b} by a right shift, and give the hexadecimal exponent of a to obtain 2^y .
6. Computations are carried out in fixed point to insure accuracy.

The effect of an argument error is $\epsilon \sim \Delta$. Since $\Delta = \delta \cdot x$, for the larger value of x , even the round-off error of the argument causes a substantial relative error in the answer.

CALLING SEQUENCE: Out-of-line.

COS Subprogram

PURPOSE: To compute the trigonometric sine (SIN) or the trigonometric cosine (COS) of a real argument representing an angle (in radians).

ENTRY POINTS: COS and SIN

RANGE: $|x| < 2^{18} \cdot \pi$

ACCURACY: The accuracy of the COS subprogram is shown in Figures 49 and 50.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
SIN		
$ x \leq \frac{\pi}{2}$	2.02×10^{-7}	1.59×10^{-6}
These statistics are based on a uniformly distributed argument sample.		

Figure 49. COS Subprogram, Relative Error

ARGUMENT RANGE	$\sigma(E)$	$M(E)$
	ROOT-MEAN-SQUARE Absolute ERROR	MAXIMUM Absolute ERROR
SIN		
$ x \leq \frac{\pi}{2}$	5.55×10^{-8}	1.31×10^{-7}
$\frac{\pi}{2} < x \leq 10$	5.53×10^{-8}	1.41×10^{-7}
$10 < x \leq 100$	5.61×10^{-8}	1.46×10^{-7}
COS		
$0 \leq x \leq \pi$	5.48×10^{-8}	1.47×10^{-7}
$-10 \leq x < 0, \pi < x \leq 10$	5.67×10^{-8}	1.42×10^{-7}
$10 < x \leq 100$	5.61×10^{-8}	1.35×10^{-7}
These statistics are based on a uniformly distributed argument sample.		

Figure 50. COS Subprogram, Absolute Error

STORAGE REQUIREMENTS: 252 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. Define $z = \frac{4}{\pi} \cdot |x|$. Separate z into the integer part q and the fraction

part r . $z = q + r$. $|x| = \frac{\pi}{4} \cdot q + \frac{\pi}{4}r$. A

long form multiplication is used to obtain accuracy.

2. If cosine is desired, add 2 to q . If sine is desired and if x is negative, add 4 to q .

This reduces the general case to the computation of $\sin(x)$ for $x \geq 0$, since

$$\cos(+x) = \sin\left(\frac{\pi}{2} + x\right)$$

$$\sin(-x) = \sin(\pi + x).$$

3. Let $q_0 \equiv q \bmod 8$.
Then for $q_0 = 0$, $\sin(x) = \sin\left(\frac{\pi}{4} \cdot r\right)$
for $q_0 = 1$, $\sin(x) = \cos\left(\frac{\pi}{4}(1-r)\right)$
for $q_0 = 2$, $\sin(x) = \cos\left(\frac{\pi}{4}r\right)$
for $q_0 = 3$, $\sin(x) = \sin\left(\frac{\pi}{4}(1-r)\right)$
for $q_0 = 4$, $\sin(x) = -\sin\left(\frac{\pi}{4} \cdot r\right)$
for $q_0 = 5$, $\sin(x) = -\cos\left(\frac{\pi}{4}(1-r)\right)$
for $q_0 = 6$, $\sin(x) = -\cos\left(\frac{\pi}{4}r\right)$
for $q_0 = 7$, $\sin(x) = -\sin\left(\frac{\pi}{4}(1-r)\right)$

These formulae reduce the case to the computation of $\left(\sin \frac{\pi}{4}r_1\right)$ or $\cos\left(\frac{\pi}{4}r_1\right)$ where $r_1 = r$ or $1-r$. $0 \leq r_1 \leq 1$.

4. $\sin\left(\frac{\pi}{4}r_1\right)$ and $\cos\left(\frac{\pi}{4}r_1\right)$ are computed using the Chebyshev interpolation polynomials of degree 3 in r^2 for the respective functions. The maximum relative error of the sine polynomial is $2^{-28.1}$ and that of the cosine polynomial is $2^{-24.6}$.

The effect of an argument error is $E \sim \Delta$. As the argument gets larger, Δ grows, and since the function value is periodically diminishing, no consistent relative error control can be maintained outside the principal range:

$$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right). \text{ Similar observation}$$

holds true for cosine as well.

CALLING SEQUENCE: Out-of-line.

DLOG Subprogram

PURPOSE: To compute the natural (DLOG) or the common (DLOG10) logarithm of a double-precision number.

ENTRY POINTS: DLOG and DLOG10

RANGE: $0 < x$

ACCURACY: The accuracy of the DLOG Subprogram is shown in Figures 51 and 52.

ARGUMENT RANGE	$\sigma(E)$ ROOT-MEAN-SQUARE	$M(E)$ MAXIMUM
	Absolute ERROR	Absolute ERROR
DLOG		
$0.5 \leq x \leq 1.5$	7.29×10^{-17}	1.85×10^{-16}
DLOG10		
$0.5 \leq x \leq 1.5$	3.09×10^{-17}	8.23×10^{-17}
These statistics are based on a uniformly distributed argument sample.		

Figure 51. DLOG Subprogram, Absolute Error

ARGUMENT RANGE	$\sigma(\epsilon)$ ROOT-MEAN-SQUARE	$M(\epsilon)$ MAXIMUM
	Relative ERROR	Relative ERROR
DLOG		
x outside (0.5, 1.5)	5.46×10^{-17}	3.31×10^{-16}
DLOG10		
x outside (0.5, 1.5)	9.96×10^{-17}	6.14×10^{-16}
These statistics are based on an exponentially distributed argument sample.		

Figure 52. DLOG Subprogram, Relative Error

STORAGE REQUIREMENTS: 365 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. Write $x = 16^p \cdot 2^{-q} \cdot m$ where p is the exponent, $0 \leq q \leq 3$ and $\frac{1}{2} \leq m < 1$.
2. Define 2 constants a, b ($a =$ base point, $2^{-b} = a$) as follows:

If $\frac{1}{2} \leq m < \frac{1}{\sqrt{2}}$, $a = \frac{1}{2}$, $b = 1$.

If $\frac{1}{\sqrt{2}} \leq m < 1$, $a = 1$, $b = 0$.

Obtain $z = \frac{m-a}{m+a}$. Then $m = a \cdot \frac{1+z}{1-z}$ and

$$|z| < 0.1716$$

$$3. \text{ Now } x = 2^{4p-q-b} \left(\frac{1+z}{1-z} \right)$$

$$\text{Hence } \log_e x = (4p-q-b) \log_e 2 + \log_e$$

$$\left(\frac{1+z}{1-z} \right)$$

4. $\log_e \left(\frac{1+z}{1-z} \right)$ is computed by using the

Chebyshev interpolation polynomial of degree 7 in z^2 for the range $0 \leq z^2 \leq 0.02944$. The maximum relative error of this polynomial is 2^{-59} .

5. If the common logarithm is wanted, use $\log_{10} x = \log_{10} e \cdot \log_e x$.

The effect of an argument error is $E \sim \delta$. This means that if the argument is close to 1, the relative error can be very large, since the function value there is very small.

CALLING SEQUENCE: Out-of-line.

DSQRT Subprogram

PURPOSE: To compute the square root of a double-precision number.

ENTRY POINT: DSQRT

RANGE: $0 \leq x$.

ACCURACY: The accuracy of the DSQRT Subprogram is shown in Figure 53.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
The full range	2.17×10^{-17}	1.08×10^{-16}
These statistics are based on an exponentially distributed argument sample.		

Figure 53. DSQRT Subprogram, Relative Error

STORAGE REQUIREMENTS: 144 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. If $x = 0$, the answer is 0.

2. Write $x = 16^{2p-q} \cdot m$, where $2p-q$ is the exponent, $q = 0$ or 1 , and m is the mantissa, $\frac{1}{16} \leq m < 1$. Then $x = 16^p \cdot 2^{-2q} \cdot \sqrt{m}$.

3. Construct the 1st approximation of \sqrt{x} as follows: $y_0 = 2^{-2q} \cdot 16^p \cdot \left(\frac{2}{9} + \frac{8}{9} m \right)$. Multiplication of 2^{-2} for the odd characteristic case is accomplished by the use of the halve instruction. The maximum relative error of this approximation is $\frac{1}{9}$.

4. Apply Newton-Raphson iteration, $y_{n+1} = \frac{1}{2} \left(y_n + \frac{x}{y_n} \right)$, 4 times to y_0 as follows:
a. Twice in the short form, and then
b. Twice in the long form.

The last step is performed as $y_4 = y_3 + \frac{1}{2} \left(\frac{x}{y_3} - y_3 \right)$ to minimize the computational truncation error. The maximum relative error of the final result is theoretically $2^{-65.7}$.

The effect of an argument error is $E \sim \frac{1}{2} \delta$.

CALLING SEQUENCE: Out-of-line.

DATAN Subprogram

PURPOSE: To compute the principal value (in radians) of the arctangent of a double-precision number.

ENTRY POINT: DATAN

RANGE: Any size double-precision argument is acceptable to this subprogram.

ACCURACY: The accuracy of the DATAN Subprogram is shown in Figure 54.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN- SQUARE Relative ERROR	MAXIMUM Relative ERROR
The full range	6.64×10^{-17}	2.08×10^{-16}
These statistics are based on tangents of uniformly distributed numbers between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.		

Figure 54. DATANH Subprogram, Relative Error

STORAGE REQUIREMENTS: 312 bytes.

METHOD:

1. Reduce the general case to the case $0 \leq x \leq 1$ by using $\text{atan}(-x) =$

$$-\text{atan}(x), \text{atan} \frac{1}{|x|} = \frac{\pi}{2} - \text{atan} |x|.$$

2. Reduce further to the case $|x| \leq \tan$

$$15^\circ \text{ by } \text{atan}(x) = 30^\circ + \text{atan} \left(\frac{\sqrt{3}x-1}{x+\sqrt{3}} \right)$$

Extra care is taken to avoid a loss of significant digits in computing $\sqrt{3}x - 1$.

3. For the basic range $|x| \leq \tan 15^\circ$, use a continued fraction of the form:

$$\frac{\text{atan } x}{x} = 1 + \frac{a_1 x^2}{b_1 + x^2} + \frac{a_2}{b_2 + x^2} + \frac{a_3}{b_3 + x^2} + \frac{a_4}{b_4 + x^2} \quad (*)$$

The coefficients were derived by transforming the continued fraction:

$$\frac{\text{atan } x}{x} = 1 + \frac{-\frac{1}{3}}{5+x^2} - \frac{\frac{3 \cdot 4}{25 \cdot 7}}{5 \cdot 9 + x^2} - \frac{\frac{16 \cdot 25}{7 \cdot 81 \cdot 11}}{9 \cdot 13} - \frac{\frac{4 \cdot 3 \cdot 49}{5 \cdot 11 \cdot 169}}{13 \cdot 17} - w$$

Here take the approximation $\frac{2}{5 \cdot 11 \cdot 13 \cdot 17}$

$(-x^{-2} + 40)$ for the value of w where the true w is:

$$w(x) = \frac{64 \cdot 27}{5 \cdot 289 \cdot 19} - \frac{179}{3 \cdot 7 \cdot 17} + x^{-2} - \text{further terms}$$

The relative error of the formula (*) is less than $2^{-57.9}$.

The effect of an argument error is $E \sim \Delta / (1+x^2)$. For small x , $\epsilon \sim \delta$; and as x becomes large, the effect of δ on ϵ diminishes.

CALLING SEQUENCE: Out-of-line.

DANH Subprogram

PURPOSE: To compute the hyperbolic tangent of a double-precision number.

ENTRY POINT: DANH

RANGE: Any size double precision argument is acceptable to this subprogram.

ACCURACY: The accuracy of the DANH Subprogram is shown in Figure 55.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN- SQUARE Relative ERROR	MAXIMUM Relative ERROR
$ x \leq 0.54931$	4.45×10^{-17}	2.00×10^{-16}
$0.54931 < x \leq 5$	2.54×10^{-17}	1.99×10^{-16}
These statistics are based on a uniformly distributed argument sample.		

Figure 55. DANH Subprogram, Relative Error

STORAGE REQUIREMENTS: 332 bytes.

CONSIDERATIONS: The subprogram DEXP is used by this subprogram.

METHOD

1. For $|x| < 0.54931$, use the following fractional approximation: $\frac{\tanh(x)}{x} =$

$$1 - \frac{a_1 x^2}{b_0 + b_1 x^2} + \frac{a_2 x^4}{b_2 x^4 + b_3 x^6} + \frac{x^8}{x^8} \quad (*)$$

where $a_1 = 676440.765$ $b_0 = 2029322.295$

$$\begin{aligned} a_2 &= 45092.124 & b_1 &= 947005.29 \\ a_3 &= 594.459 & b_2 &= 52028.55 \\ & & b_3 &= 630.476 \end{aligned}$$

This formula was obtained by transforming the continued fraction,

$$\frac{\tanh(x)}{x} = \frac{1}{1 + \frac{x^2}{3 + \frac{x^2}{5 + \dots + \frac{x^2}{15 + \frac{x^2}{w}}}}}$$

with an approximate value 0.017 for

$$w = \frac{x^2}{17 + \frac{x^2}{19 + \dots}}$$

The maximum relative error of the formula (*) is $2^{-64.5}$.

2. For $0.54931 \leq x \leq 20.101$, use $\tanh(x) = 1 - \frac{2}{e^{2x} + 1}$.
3. For $20.101 \leq x$, use $\tanh(x) \cong 1$.
4. For $x \leq -0.54931$, $\tanh(x) = -\tanh(-x)$.
5. The long form exponential function is used in the case 2 above.

The effect of an argument error is $E \sim (1 - \tanh^2 x) \Delta$, $\epsilon \sim 2\Delta / \sinh 2x$. Thus, for small x , $\epsilon \sim \delta$, and as x gets larger, the effect of δ on ϵ diminishes.

CALLING SEQUENCE: Out-of-line.

DCOS Subprogram

PURPOSE: To compute the sine (DSIN) or cosine (DCOS) of a double-precision argument representing an angle (in radians).

ENTRY POINTS: DCOS and DSIN

RANGE: $|x| < 2^{50} \cdot \pi$

ACCURACY: The accuracy of the DCOS Subprogram is shown in Figures 56 and 57.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
DSIN		
$ x \leq \frac{\pi}{2}$	4.85×10^{-17}	4.08×10^{-16}
These statistics are based on a uniformly distributed argument sample.		

Figure 56. DCOS Subprogram, Relative Error

ARGUMENT RANGE	$\sigma(E)$	$M(E)$
	ROOT-MEAN-SQUARE Absolute ERROR	MAXIMUM Absolute ERROR
DSIN		
$ x \leq \frac{\pi}{2}$	2.17×10^{-17}	9.10×10^{-17}
$\frac{\pi}{2} < x \leq 10$	6.35×10^{-17}	1.64×10^{-16}
$10 < x \leq 100$	1.03×10^{-15}	2.69×10^{-15}
DCOS		
$0 \leq x \leq \pi$	6.40×10^{-17}	1.79×10^{-16}
$-10 \leq x < 0, \pi < x \leq 10$	5.93×10^{-17}	1.76×10^{-16}
$10 < x \leq 100$	1.01×10^{-15}	2.65×10^{-15}
These statistics are based on a uniformly distributed argument sample.		

Figure 57. DCOS Subprogram, Absolute Error

STORAGE REQUIREMENTS: 368 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. Divide $|x|$ by $\frac{\pi}{4}$ and decompose the quotient into the integer part and fraction part.

$$y = |x| \cdot \frac{4}{\pi} = q + r, \quad q \text{ integer}, \quad 0 \leq r < 1.$$

2. If cosine entry, add 2 to q . If sine entry with negative argument add 4 to q . Let $q_0 \equiv q \bmod 8$.

$$\cos(x) = \sin\left(|x| + \frac{\pi}{2}\right),$$

$$\sin(-x) = \sin(|x| + \pi).$$

3. Now the answer is

$$\sin \frac{\pi}{4} (q_0 + r) \quad 0 \leq q_0 \leq 7$$

$$\text{Compute } \sin \frac{\pi}{4} r \quad \text{if } q_0 = 0 \text{ or } 4$$

$$\cos \frac{\pi}{4} (1-r) \quad \text{if } q_0 = 1 \text{ or } 5$$

$$\cos \frac{\pi}{4} r \quad \text{if } q_0 = 2 \text{ or } 6$$

$$\sin \frac{\pi}{4} (1-r) \quad \text{if } q_0 = 3 \text{ or } 7$$

$$\frac{1}{r_0} \sin \frac{\pi}{4} r_0, \text{ where } r_0 \text{ is } r \text{ or } 1-r, \text{ is}$$

computed by the use of the Chebyshev interpolation polynomial of degree 6 in r_0^2 in the range $0 \leq r_0^2 \leq 1$. The maximum relative error of this polynomial is 2^{-58} .

$\cos \frac{\pi}{4} r_0$ is computed

by using the Chebyshev interpolation polynomial of degree 7 in r_0^2 in the range $0 \leq r_0^2 \leq 1$. The maximum relative error of this polynomial is $2^{-64.3}$.

4. If $q \leq 4$, give negative sign to the result.

The effect of an argument error is $E \sim \Delta$. As the argument gets larger, Δ grows, and since the function value is periodically diminishing, no consistent relative error control can be maintained

outside the principal range $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. This holds for both sine and cosine.

CALLING SEQUENCE: Out-of-line.

DEXP Subprogram

PURPOSE: To compute the value of "e" raised to the power of a double-precision argument.

ENTRY POINT: DEXP

RANGE: $x < 174.67309$

ACCURACY: The accuracy of the DEXP subprogram is shown in Figure 58.

ARGUMENT RANGE	$\sigma(\epsilon)$	$M(\epsilon)$
	ROOT-MEAN-SQUARE Relative ERROR	MAXIMUM Relative ERROR
$ x \leq 1$	7.49×10^{-17}	2.27×10^{-16}
$1 < x \leq 20$	8.69×10^{-16}	2.31×10^{-15}
$20 < x \leq 170$	9.33×10^{-16}	2.33×10^{-15}
These statistics are based on a uniformly distributed argument sample.		

Figure 58. DEXP Subprogram, Relative Error

STORAGE REQUIREMENTS: 452 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the argument is within the valid argument range; if it is not, an error message is printed and execution is terminated.

METHOD:

1. If $x < -180.2183$, give 0 as the answer.
2. Divide x by $\log_e 2$ and decompose the quotient as:

$$y = x / \log_e 2 = 4a - b - \frac{c}{16} - d$$
 where a , b , and c are integers,
 $0 \leq b \leq 3$, $0 \leq c \leq 15$ and $0 \leq d < \frac{1}{16}$.
 Then $e^x = 2^y = 16^a \cdot 2^{-b} \cdot 2^{-c/16} \cdot 2^{-d}$.
3. Compute 2^{-d} by using the Chebyshev interpolation polynomial of degree 6 over the range $0 \leq d < \frac{1}{16}$. The maximum relative error of this polynomial is 2^{-57} .
4. If $c > 0$, multiply 2^{-d} by $2^{-c/16}$. The 15 constants $2^{-c/16}$, $1 \leq c \leq 15$ are included in the subroutine.
5. If $b > 0$, halve the result b - times.
6. Multiply by 16^a by adding a to the characteristic of the result.

The effect of an argument error is $\epsilon \sim \Delta$. Since $\Delta = \epsilon \cdot x$, for the larger value of x , even the round-off error of the argument causes substantial relative error in the result.

CALLING SEQUENCE: Out-of-line.

MOD Subprogram

PURPOSE: To compute the result of the first integer argument modulo the second integer argument. Modulo is a mathematical operator which yields the remainder function of division. Thus, 9 modulo 6 \equiv 3.

ENTRY POINT: MOD

RANGE: Any arguments, except as noted under considerations, are acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 58 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the second argument is not zero; if it is zero, the first argument is given as the result.

METHOD: $\text{MOD}(x,y) = x - (x/y) * y$

The result is calculated according to the above formula. If this result is negative, the absolute value of modulus y is added to it, giving a non-negative value less than y.

CALLING SEQUENCE: Out-of-line.

AMOD Subprogram

PURPOSE: To compute the result of the first real-number argument (AMOD) or double-precision argument (DMOD) modulo the second argument. Modulo is a mathematical operator which yields the remainder function of division. Thus $3 \equiv 39 \text{ modulo } 6$.

ENTRY POINTS: AMOD and DMOD.

RANGE: Any arguments, except as noted under considerations, are acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 112 bytes.

CONSIDERATIONS: Checking is done at object time to ensure that the second argument is not zero; if it is zero the first argument is given as the result.

METHOD: $\text{AMOD or DMOD}(x,y) = x - (x/y) * y$

The result is calculated according to the above formula. If this result is negative, the absolute value of modulus y is added to it, giving a non-negative value less than y.

CALLING SEQUENCE: Out-of-line.

MAX0 Subprogram

PURPOSE: To select the maximum (AMAX0 or MAX0) or the minimum (AMIN0 or MIN0) integer value from a list of integer numbers, with the option of converting the result (AMAX0 or AMIN0) to a real number or giving an integer result (MAX0 or MIN0).

ENTRY POINTS: MAX0, AMAX0, MIN0, and AMIN0

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 185 bytes.

CONSIDERATIONS: None.

METHOD: Starting with the second argument and proceeding to the end of the argument list, each argument is algebraically compared to the maximum (AMAX0 and MAX0) or to the minimum (AMIN0 and MIN0) of the previous arguments to yield a new maximum or minimum. For AMAX0 or AMIN0 the result is then converted to a real number.

CALLING SEQUENCE: Out-of-line.

MAX1 Subprogram

PURPOSE: To select the maximum (AMAX1 or MAX1) or the minimum (AMIN1 or MIN1) real value from a list of real numbers, with the option of converting the result (MAX1 or MIN1) to an integer number or giving a real-number result (AMAX1 or AMIN1).

ENTRY POINTS: MAX1, AMAX1, MIN1, and AMIN1

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 185 bytes.

CONSIDERATIONS: None.

METHOD: Starting with the second argument and proceeding to the end of the argument list, each argument is algebraically compared to the maximum (AMAX1 and MAX1) or to the minimum (AMIN1 and MIN1) of the previous arguments to yield a new maximum or minimum. For MAX1 or MIN1 the result is then converted to an integer number.

CALLING SEQUENCE: Out-of-line.

DMAX1 Subprogram

PURPOSE: To select the maximum (DMAX1) or the minimum (DMIN1) double-precision value from a list of double-precision numbers.

ENTRY POINTS: DMAX1 and DIN1

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 101 bytes.

CONSIDERATIONS: None.

METHOD: Starting with the second argument and proceeding to the end of the argument list, each argument is algebraically compared to the maximum (DMAX1) or to the minimum (DMIN1) of the previous arguments to yield a new maximum or minimum.

CALLING SEQUENCE: Out-of-line.

FLOAT Subprogram

PURPOSE: To convert an integer number to a single-precision real number (FLOAT) or a double-precision real number (DFLOAT).

ENTRY POINTS: FLOAT and DFLOAT

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: There is a loss of precision when absolute x is greater than $2^{*}24$.

STORAGE REQUIREMENTS: 80 bytes.

CONSIDERATIONS: If the argument exceeds the valid argument range the result will be meaningless.

METHOD: The absolute value of the argument is used as the mantissa, a proper characteristic is created, and the sign of the argument is transferred to the result.

CALLING SEQUENCE: In-line.

IFIX Subprogram

PURPOSE: To convert a single- real number to an integer number (IFIX) and to truncate the fractional portion of the mantissa of a real number and convert the modified real value to an integer number.

ENTRY POINTS: IFIX, INT and IDINT.

RANGE: $|x| \leq (2^{*}31)$

ACCURACY: There is a loss of precision when absolute x is greater than $2^{*}24$.

STORAGE REQUIREMENTS: 112 bytes.

CONSIDERATIONS:

1. The library subprogram IFIX is used to convert the modified value.
2. If the value exceeds the valid argument range, the result is given as zero.

METHOD: An unnormalized add of floating point zero is used to shift the fractional part of the absolute value of the argument out of the register. The exponent is discarded and the sign of the argument is transferred to the result.

CALLING SEQUENCE: In-line.

AINT SUBPROGRAM

PURPOSE: To truncate the fractional portion of the mantissa of a real number.

ENTRY POINT: AINT

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 68 bytes.

CONSIDERATIONS: If the absolute value of the argument is less than one the result is zero.

METHOD: AINT (x) = sign of x times the largest integer $\geq |x|$.

The fractional portion of the argument is deleted.

CALLING SEQUENCE: Out-of-line.

ABS Subprogram

PURPOSE: To obtain the absolute value of a real number.

ENTRY POINT: ABS

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 2-6 bytes.

CONSIDERATIONS: The argument may be a mixed-mode expression but the result of the expression must be the same type as the function.

METHOD: ABS (x) = $|x|$

The sign of x is forced to plus.

CALLING SEQUENCE: In-line.

DABS Subprogram

PURPOSE: To obtain the absolute value of a double-precision number.

ENTRY POINT: DABS

RANGE: Any size argument is acceptable to this subprogram.

Accuracy: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 2-6 bytes.

CONSIDERATIONS: The argument may be a mixed-mode expression but the result of the expression must be the same type as the function.

METHOD: DABS $(x) = |x|$

The sign of x is forced to plus.

CALLING SEQUENCE: In-line.

DIM Subprogram

PURPOSE: To compute the result of the first real argument minus the smaller of the first and second real arguments.

ENTRY POINT: DIM

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 8-18 bytes.

CONSIDERATIONS: None.

METHOD: DIM $(x,y) = x - \text{minimum}(x,y)$

y is subtracted from x , if the difference is negative the result is equal to zero. If the difference is positive the result is equal to the difference.

CALLING SEQUENCE: In-line.

DSIGN Subprogram

PURPOSE: To transfer the sign of the second double-precision argument to the first double-precision argument.

ENTRY POINT: DSIGN

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 10-20 bytes.

CONSIDERATIONS: None.

METHOD: DSIGN (x,y) sign of $y * x$

The value of y is tested. If positive, the sign of x is forced plus; if negative the sign of x is forced minus.

CALLING SEQUENCE: In-line.

IABS Subprogram

PURPOSE: To obtain the absolute value of an integer number.

ENTRY POINT: IABS

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: The only error produced is when the argument is -2^{31} .

STORAGE REQUIREMENTS: 2-6 bytes.

CONSIDERATIONS: The argument may be a mixed-mode expression but the result of the expression must be the same type as the function.

METHOD: IABS $(x) = |x|$

The sign of x is forced to plus.

CALLING SEQUENCE: In-line.

IDIM Subprogram

PURPOSE: To compute the result of the first integer argument minus the smaller of the first and second integer arguments.

ENTRY POINT: IDIM

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 8-18 bytes.

CONSIDERATIONS: None.

METHOD: IDIM (x, y) = x - minimum (x, y)

y is subtracted from x, if the difference is negative the result is equal to zero. If the difference is positive the result is equal to the difference.

CALLING SEQUENCE: In-line.

ISIGN Subprogram

PURPOSE: To transfer the sign of the second integer argument to the first integer argument.

ENTRY POINT: ISIGN

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by this subprogram.

STORAGE REQUIREMENTS: 10-20 bytes.

CONSIDERATIONS: None.

METHOD: ISIGN (x,y) = sign of y*|x|

The value of y is tested. If positive, the sign of x is forced plus; if negative the sign of x is forced minus.

CALLING SEQUENCE: In-line.

SIGN Subprogram

PURPOSE: To transfer the sign of the second real number argument to the first real number argument.

ENTRY POINT: SIGN

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 10-20 bytes.

CONSIDERATIONS: None.

METHOD: SIGN (x,y) = sign of y*|x|

The value of y is tested. If positive, the sign of x is forced plus; if negative the sign of x is forced minus.

CALLING SEQUENCE: In-line.

SNGL Subprogram

PURPOSE: To convert a double-precision argument to a real argument.

ENTRY POINT: SNGL

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: None.

CONSIDERATIONS: None.

METHOD: The most significant part of a double-precision number is supplied.

CALLING SEQUENCE: In-line.

DBLE Subprogram

PURPOSE: To convert a real argument to a double-precision argument.

ENTRY POINT: DBLE

RANGE: Any size argument is acceptable to this subprogram.

ACCURACY: No error is produced by the computation in this subprogram.

STORAGE REQUIREMENTS: 4-6 bytes.

CONSIDERATIONS: None.

METHOD: The least significant part of a double precision number is supplied.

CALLING SEQUENCE: In-line.

SERVICE SUBPROGRAM DESCRIPTIONS

The FORTRAN service subprograms are all out-of-line subprograms. The description of each service subprogram includes:

1. Purpose.
2. Permissible entry points.
3. Format.
4. Storage requirements.
5. Considerations that should be noted in using the subprogram.

6. Usage.

In the following descriptions, i represents an integer expression and j represents an integer variable.

EXIT Subprogram

PURPOSE: To terminate the execution of a program and returns control to the system director. Entry Point: EXIT

FORMAT: CALL EXIT

STORAGE REQUIREMENTS: 24 bytes.

CONSIDERATIONS: This subprogram performs the same function as the STOP statement. A program written in assembler language may use the EXIT subprogram; a program written in FORTRAN may use either the STOP statement or the EXIT subprogram.

USAGE: Control is given to a routine which performs internal services and returns control to the FORTRAN system director.

DUMP Subprogram

PURPOSE: To dump the indicated limits of storage in the specified format with (DUMP) or without (PDUMP) program termination.

ENTRY POINTS: DUMP and PDUMP

FORMAT: CALL DUMP or CALL PDUMP
(A₁, B₁, F₁, ..., A_n, B_n, F_n)

Where A and B are variable data names indicating the limits of storage to be dumped; either A or B may represent the upper or lower limits of storage to be dumped; F is an integer that indicates the format of the dump.

STORAGE REQUIREMENTS: 445 bytes.

CONSIDERATIONS: None.

USAGE: Each set of A, B, and F parameters are treated separately. If no parameters are given, all storage is dumped in hexadecimal. The A and B parameters are examined to determine which indicates the lower limit and which indicates the upper limit of storage to be dumped.

The value of F is used to determine the format of the dump as follows:

- 0 specifies a hexadecimal dump format;
- 4 specifies an integer dump format;

- 5 specifies a real dump format;
- 6 specifies a double-precision dump format.

If F is not specified, the dump is in hexadecimal format. The appropriate conversion routine is then called to format the records to be dumped and these records are written by the input/output routine.

SLITE Subprogram

PURPOSE: To turn sense lights on or off (SLITE), and test whether a sense light is on or off (SLITET).

ENTRY POINTS: SLITE and SLITET

STORAGE REQUIREMENTS: 178 bytes. This is the combined size of subprogram SLITE and SLITET.

SLITE SUBPROGRAM: This subprogram turns sense lights on or off.

FORMAT: SLITE (i) where i is 0, 1, 2, 3, or 4.

CONSIDERATIONS: If the value of i is not 0, 1, 2, 3, or 4 a message is printed and execution is terminated.

USAGE: The value of i is tested and the corresponding sense light turned on; if the value of i is zero, all sense lights are turned off. Each sense light occupies one byte of storage. When the byte contains zeros it is off; when it does not contain zeros it is on.

SLITET SUBPROGRAM: This subprogram tests whether a sense light is on or off.

FORMAT: SLITET (i,j), where i is 1, 2, 3, or 4; j is set to 1 or 2.

CONSIDERATIONS: If the value of i is not 1, 2, 3, or 4 a message is printed and execution is terminated.

USAGE: The value of i is tested and the corresponding sense light is examined; j is then set to 1 if the sense light was on or 2 if the sense light was off. The sense light that was tested is then turned off. Each sense light occupies one byte of storage. When the byte contains zeros it is off; when it does not contain zeros it is on.

OVERFL Subprogram

PURPOSE: To test for exponent overflow or underflow.

ENTRY POINT: OVERFL

FORMAT: OVERFL (j) Where j is set to 1, 2, or 3.

STORAGE REQUIREMENTS: 76 bytes.

CONSIDERATIONS: None.

USAGE: The status of the overflow indicator is examined; j is set to 1 if a floating-point overflow condition exists, that is if the result of an arithmetic operation is greater than 16^{63} ; j is set to 2 if no overflow condition exists; j is set to 3 if a floating-point underflow condition exists, that is if the result of an arithmetic operation is not zero but less than 16^{-63} . After j is set the machine is left in a no overflow condition.

DVCHK Subprogram

PURPOSE: To test for divide check interruptions.

ENTRY POINT: DVCHK

FORMAT: DVCHK (j) where j is set to 1 or 2.

STORAGE REQUIREMENTS: 60 bytes.

CONSIDERATIONS: None.

USAGE: The status of the divide check indicator is examined; j is set to 1 if the indicator is on or 2 if the indicator is off. After j is set, the indicator is turned off. The divide check indicator occupies one byte of storage. When the byte contains zeros it is off; when it contains any other bit pattern it is on.

APPENDIX E: COMPILER CARD OUTPUT

The Basic Programming Support FORTRAN IV compiler generates seven types of object program statements (on cards or tape) when compiling a program. They are: 12-2-9 ESD (three types), 12-2-9 TXT, 12-2-9 RLD, 12-2-9 and END. The format and description of each statement is given in the following text.

ESD TYPE 0 STATEMENT

The External Symbol Dictionary (ESD) Type 0 statement defines the name and relative starting address of a compiled program. There is only one ESD Type 0 card for each compiled program in a job. The format of the ESD Type 0 statement is explained in Figure 59 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code 12-0-1-8-9 and 12-11-1-8-9 (hexadecimal value of 0010).
13-14	Blank.
15-16	External Symbol Identification (ESID). Number, in extended card code, assigned to the segment.
17-22	Program name.
23	Blank.

(continued)

Column	Contents
24	Extended card code 12-3-8 (hexadecimal value of a period character) used by the loader in conjunction with the program name.
25	Extended card code 12-0-1-8-9 (hexadecimal value of 00), identifies this as a program name card.
26-28	Address, in extended card code, of the first byte of the program as assigned by the compiler.
29	Blank.
30-32	Number of bytes in the program. This field is always punched in extended card code with the hexadecimal value of 000000.
33-72	Blank.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with a sequential number of 0001.

Figure 59. ESD Type 0 Statement

ESD TYPE 1 STATEMENT

The External Symbol Dictionary (ESD) Type 1 statement defines an entry point name within the program to which another program may refer. One card is always produced for each compiled program.

The format of the ESD Type 1 statement is explained in Figure 60 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code (hexadecimal value of 0010).
13-16	Blank.
17-22	Name of entry point.
23-24	Blank.
25	Extended card code 12-1-9 (hexadecimal value of 01), identifies this as an entry point card.
26-28	The address 000008, in extended card code, of the entry point as assigned by the compiler.
29-30	Blank.
31-32	External Symbol Identification (ESID). The number 0001, in extended card code, assigned to the program in which entry points occur.
33-72	Blank.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with a sequential number of 0002.

Figure 60. ESD Type 1 Statement

ESD TYPE 2 STATEMENT

The External Symbol Dictionary (ESD) Type 2 statement points to a name within another program to which this program may refer. It is assigned an identifying number of 2 through 15, according to the order in which it is encountered among the external symbols in the program.

The format of the ESD Type 2 statement is explained in Figure 61 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code (hexadecimal value of 0010, 0020, or 0030).
13-14	Blank.
15-16	External Symbol Identification (ESID). Sequential number, in extended card code, assigned to the first external symbol on this card.
17-22	Name of external symbol.
23-24	Blank.
25	Extended card code 12-1-9 (hexadecimal value of 02) identifies this as an external symbol card.
26-28	Extended card code 12-0-1-8-9, 12-0-1-8-9, and 12-0-1-8-9 (hexadecimal value of 000000). An address of 0 is always assigned to External Symbols by the Compiler.
29-32	Blank.
33-48	This field is used for the second external symbol and has the same format as columns 17-32. (If not used, the field is blank.)
49-64	This field is used for the third external symbol and has the same format as columns 17-32. (If not used, the field is blank.)
73-76	This field is always punched with the first four characters of the program.
77-80	This field is always punched with a sequential number.

Figure 61. ESD Type 2 Statement

ESD TYPE 5 STATEMENT

The External Symbol Dictionary (ESD) Type 5 statement defines the existence and size of the Blank COMMON area reserved for the job.

The format of the ESD Type 5 statement is explained in Figure 62 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	The number of bytes in the card. Extended card code 12-0-1-8-9 and 12-11-1-8-9 (hexadecimal value of 0010).
13-14	Blank.
15-16	External Symbol Identification (ESID). The number 0002, in extended card code, assigned to the program.
17-24	Blank.
25	Extended card code 12-5-9 (hexadecimal value of 05) identifies this as a Blank COMMON card.
26-28	Address of the first byte of the program as assigned by the compiler. This field is always punched in extended card code with the hexadecimal value of 000000.
29	Blank.
30-32	Number, in extended card code, of bytes in Blank COMMON.
33-72	Blank.
73-76	This field is always punched with first four characters of the program name.
77-80	This field is always punched with a sequential number of 0003.

Figure 62. ESD Type 5 Statement

TXT STATEMENT

The Text (TXT) statement contains the instructions and/or constants of the user-written program and the starting address at which the first byte of text is to be loaded.

The format of the TXT statement is explained in Figure 63 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	TXT. Identifies the type of load card.
5	Blank.
6-8	24-bit starting address (in extended card code) in storage where the information from the card is to be loaded.
9-10	Blank.
11-12	Number of bytes (in extended card code) of text to be loaded from the card.
13-14	Blank.
15-16	External Symbol Identification (ESID). Number, in extended card code (hexadecimal value of 0001), assigned to the program in which the text occurs.
17-72	A maximum of 56 bytes of instructions and/or constants assembled in extended card code.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with a sequential number.

Figure 63. TXT Statement

RLD STATEMENT

The Relocation List Dictionary (RLD) statement is produced when the compiler encounters an address which is an internal

or external symbol. An internal symbol represents a point within the program to which control of execution can be transferred. An external symbol represents a point within the program from which control of execution can be transferred to another program. The format of the RLD statement is explained in Figure 64 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	RLD. Identifies the type of load card.
5-10	Blank.
11-12	Number, in extended card code, of bytes of information in the variable field (card columns 17-72) of this card. The range is from 8 to a maximum of 56.
13-16	Blank.
17-72	Variable field (in extended card code). Consists of the following subfields: <u>Relocation Header.</u> (Two bytes.) The ESID with a value of from 01 through 15. Whether or not the value is 01 or from 02 through 15 depends on whether the symbol it points to is internal or external to the particular program. <u>Position Header.</u> (Two bytes.) The ESID assigned to this program. <u>Flag Byte</u> (bits 0 through 3 are not used). This byte contains three items: 1. <u>Size.</u> (Bits 4 and 5.) Two bits which indicate the length (in bytes) of the adjusted address cell (AA Cell). a. 00 - one-byte cell b. 01 - two-byte cell c. 10 - three-byte cell d. 11 - four-byte cell

(continued)

Column	Contents
	2. <u>Complement Flag.</u> (Bit 6.) When this bit is a one, it means that the value (or address) of the symbol is to be subtracted from the contents of the AA Cell. When this bit is a zero, the value of the symbol is to be added to the contents of the AA Cell.
	3. <u>Continuation Flag.</u> (Bit 7.) When this bit is a one, it means that this is one of a series of addresses to be adjusted. When this bit is a zero, this is the only AA Cell to be adjusted or the last in a series using the same Relocation and Position headers.
	<u>Address.</u> The three-byte address of the location of the AA Cell. The Flag Byte and Address may be repeated for AA Cells as long as the continuation flag bit is on in the current four-byte entry.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with a sequential number.

Figure 64. RLD Statement

END STATEMENT

The END statement is produced by the compiler when it encounters an END source program statement. This statement ends the loading of a compiled program and may specify a location within the program to which control is transferred at end-of-load.

The format of the END statement is explained in Figure 65 in terms of an 80-column card.

Column	Contents
1	Load card identification (12-2-9). Identifies this as a card acceptable to the loader.
2-4	END. Identifies the type of load card.
5	Blank.
6-8	Address (may be blank), in extended card code, of the point in the program to which control may be transferred at the end of the loading process.
9-14	Blank.
15-16	External Symbol Identification (ESID). The number 0001, in extended card code, is assigned to the program.
17-28	Blank.
29-32	Number, in extended card code, of bytes in the program.
33-71	Blank.
73-76	This field is always punched with the first four characters of the program name.
77-80	This field is always punched with the last sequential number.

Figure 65. END Statement

This section describes the procedures for changing the Device Assignment Table (DAT) to correspond with the I/O devices available at a particular installation.

The DAT supplied by IBM contains data set reference number assignments for a 2540 Card Read Punch with an actual address of 00C. If a user is equipped with a card read punch of which the actual address is not 00C, input cards cannot be read until the DAT is initially changed to reflect this.

The process for changing the DAT is as follows:

1. Press the Load Key. This causes the FORTRAN system director (FSD) to be loaded into storage, along with the DAT and control card routine. When control of execution is passed to the control card routine, it attempts to read from the device associated with the address 00C. If a card read punch is not associated with the address 00C, the message:

FIS 00C

is printed signifying an I/O error condition. The system then enters a wait state.

2. Using the Load Address Switch, address the card read punch.
3. Punch the information shown in Figure 66 on an 80-column card. This card represents a program which, when executed, causes the next card in the card read punch to be read.

Column	Multiple Punch
1	12-0-1-8-9
2	12-0-1-8-9
3	12-0-1-8-9
4	12-0-1-8-9
5	12-0-1-8-9
6	12-0-1-8-9
7	0-5-9
8	12-0-1-8-9
9	12-2-9
10	12-0-1-8-9
11	12-0-1-8-9
12	12-0-8
13	11-0-1-8-9
14	12-0-1-8-9
15	12-0-1-8-9
16	12
17-80	Blank

Figure 66. Program Card Layout

4. Punch a /SET card to change the DAT on an 80-column card as follows:

Column 1	Column 10
↑	↑
/SET	1=adr (type)

where adr is the actual hexadecimal address of the card read punch and type is 2540 or 1442.

5. Place the program card and /SET card, in that order, in the card read punch.
6. Press the Load Key. This causes the first eight columns of the program card to be placed into the Program Status Word (PSW); the second eight columns to be placed into the Channel Command Word (CCW) as shown in Figure 67.

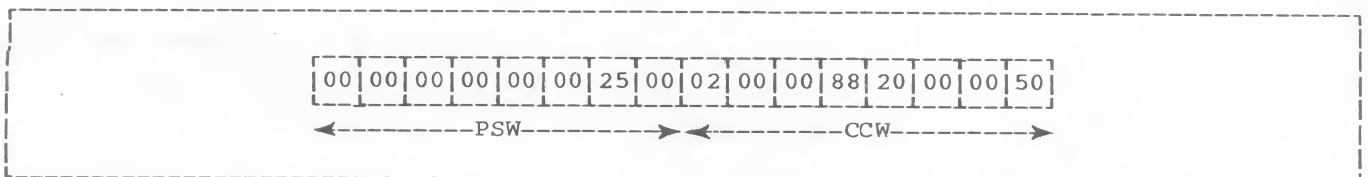


Figure 67. PSW and CCW Contents After Program Card Load

The instruction in the CCW causes the next card (i.e., the /SET card) in the card read punch to be read into location 88. The /SET statement causes the DAT to be changed to reflect the address of the card read punch being used.

7. After the initial change to the DAT is made, additional /SET control cards may be used to tailor the DAT to the I/O configuration of a particular installation. (See the section, "FORTRAN System Maintenance.")

ABS subprogram 71
 Accuracy, mathematical subprograms 59
 /AFTER control statement 28,33
 Format 34
 Library subprogram name 33
 User-written subprogram 35
 ALOG subprogram 34,60
 ALOG10 subprogram 34,60
 AMAX0 subprogram 34,70
 AMAX1 subprogram 34,70
 AMIN0 subprogram 34,70
 AMIN1 subprogram 34,70
 AMOD subprogram 34,70
 /* (asterisk) control statement 36
 ATAN subprogram 34,62

Basic programming support FORTRAN IV system 5

Bus out check 40

C

 Data Conversion 14-15
 Exponent-overflow exception 51-52
 Called and calling programs 53
 Card Read Punch
 1402 (See 2540)
 1442 23-25
 2540 24-25
 CARDS option 17
 CCW (See Channel command word)
 CGOTO subprogram 34
 Chaining check 40
 Channel command word 81-82
 Channel control check 39
 Channel data check 39
 Command reject 40
 COMMON area 58
 Compilation and execution 5,7-25
 Compilation speed considerations 22
 Compiler, basic programming support FORTRAN 7
 Complement flag 79
 Continuation flag 79
 Control statements 5-6
 Compilation and execution 5,7
 System maintenance 27
 COS subprogram 34,64

D (See Exponent-underflow exception)
 DABS subprogram 34,72
 DAT (See Device assignment option)
 /DATA control statement 12-13
 Data conversion 14-15
 Data converted check 40
 DATAN subprogram 34,66
 DBLE subprogram 73
 DCOS subprogram 34,68
 DECK option 9
 Deck structure
 Compiling and executing 17-20
 Editing 36-37

Definition of symbols, library subprograms 59
 /DELETE control statement 28,35
 Format 35
 Names of subprograms deleted 35
 Density (See Tape density)
 Describing control statements 5-6
 Device assignment
 C data conversion 14-15
 H density 14
 L density 14
 M density 14
 N data conversion 14-15
 Option 14,81
 T data conversion 14-15
 Table (DAT) 13
 Types 14
 DEXP subprogram 34,69
 DFLOAT subprogram 34,71
 Diagnostic messages 7
 Object program 20,50-52
 Source program 20,42-49
 DIM subprogram 72
 DLOG subprogram 34,65
 DLOG10 subprogram 34,65
 DMAX1 subprogram 34,70
 DMIN1 subprogram 34,70
 DMOD subprogram 34,70
 DSIGN subprogram 72
 DSIN subprogram 34,68
 DSQRT subprogram 34,66
 DTANH subprogram 34,67
 DUMP subprogram 34,74
 DVCHK subprogram 34,75

Editing the FORTRAN system 27-37
 EDR control statement 28,31-33
 End of job 39
 END statement 28,36
 As a compiler output card 79-80
 Entry point names within subprograms
 ALOG10 34,60
 AMAX0 34,70
 AMAX1 34,70
 AMIN0 34,70
 AMIN1 34,70
 DFLOAT 34,71
 DLOG10 34,65
 DMIN1 34,70
 DMOD 34,70
 DSIN 34,68
 IDINT 34,71
 INT 34,71
 MIN0 34,70
 MIN1 34,70
 PDUMP 34,74
 SIN 34,64
 SLITET 34,74
 Entry points 33-34
 Equipment check 39
 Error code diagnostics 20,50-52

ESD statement 76-78
 Type 0 76
 Type 1 76
 Type 2 77
 Type 5 78
 ESID (See External symbol identification)
 Execution (See also /LOAD control statement) 5,7
 EXIT subprogram 34,73
 EXP subprogram 34,63
 Exponent-overflow exception 51-52
 Exponent-underflow exception 51-52
 External symbol identification 76-80
 F (See Floating-point-divide exception)
 FDXPD subprogram 34
 FDXPI subprogram 34
 FIA xxx 40
 FID xxx 40
 First byte address
 12-2-9 EDR control statement 28,32
 12-2-9 REP control statement 28,30
 FIS xxx 39
 FIXDI subprogram 34
 FLOAT subprogram 34,71
 Floating-point-divide exception 51-52
 Format of control statements
 /AFTER 33-35
 /DATA 12-13
 /DELETE 35
 /EDIT 28-29
 /FTC 9-12
 /JOB 7-8
 /LOAD 16,17
 /SET 13-16,29-30
 12-2-9 EDR 31-33
 12-2-9 END 36,79-80
 12-2-9 ESD 76-78
 12-2-9 REP 30-31
 12-2-9 RLD 78-79
 12-2-9 TXT 78
 FORTRAN
 Job 7
 Language 5
 Linkage conventions 53-54
 System tape 27-28
 1402 Card Read Punch (See 2540 card read punch)
 1442 Card Read Punch 23-25
 FRXPI subprogram 34
 FRXPR subprogram 34
 /FTC control statement 9-12
 GO option 8
 GOGO option 8
 H (See Tape density)
 IABS subprogram 72
 IBERR subprogram 34
 IDIM subprogram 34,72
 IDINT subprogram 34,71
 IFIX subprogram 34,71
 In-line parameter area 58
 Input/output interruption 39
 Instruction counter (See Storage address)
 INT subprogram 34,71
 Interface control check 39

Internal statement number 10,42
 Interrupt key 39-40
 ISIGN subprogram 73
 Isn (See Internal statement number)

/JOB control statement 7-8
 Job, FORTRAN 7
 L (See Tape density)
 Library subprogram name
 /AFTER control statement 33
 /DELETE control statement 35
 LINE option 15
 LIST option 9
 Load address switch 39-40,81
 /LOAD control statement 16-17
 Load key 37,39-40,81
 Lowest level subprogram 58

M (See Tape density)
 Machine considerations 22-23
 Machine requirements 7
 Minimum 22-23
 MAP option 8,10,17
 /FTC control statement 10
 /LOAD control statement 17
 Mathematical subprogram descriptions 59
 Mathematical subprogram names 33-34
 ABS 71
 ALOG 60
 AMOD 70
 ATAN 62
 COS 64
 DABS 72
 DATAN 66
 DBLE 73
 DCOS 68
 DEXP 69
 DIM 72
 DLOG 65
 DMAX1 70
 DSIGN 72
 DSQRT 66
 DTANH 67
 EXP 63
 FLOAT 71
 IABS 72
 IDIM 72
 IFIX 71
 ISIGN 73
 MAX0 70
 MAX1 70
 MOD 69
 SIGN 73
 SNGL 73
 SQRT 61
 TANH 63

MAX0 subprogram 34,70
 MAX1 subprogram 34,70
 Messages 5
 Object program 20,50-53
 Operator 39-40
 Source program 20,42-49
 MIN0 subprogram 34,70
 MIN1 subprogram 34,70
 MOD subprogram 34,69
 MULTIPLE option 7-8

N (See Data conversion)
 NAME option 9
 NODECK option 9
 NOGO option 8
 NOLIST option 9
 NOMAP option 10,17
 /FTC control statement 10
 /LOAD control statement 16-17

 Object code replacement
 12-2-9 EDR control statement 28,31-33
 12-2-9 REP control statement 28,30-31
 Object program
 Diagnostic messages 20,50-53
 Program size 21
 Operating procedures 5
 Operator messages 39-40
 Options
 CARDS 17
 DECK 9
 Device assignment 14
 GO 8
 GOGO 8
 LINE 15
 LIST 9
 MAP 10
 MULTIPLE 7-8
 NAME 7
 NODECK 9
 NOGO 8
 NOLIST 9
 NOMAP 9
 SINGLE 7-8
 SIZE 11
 TAPE 17
 OVERFL subprogram 34,75
 Overrun 40

 Parameter area 54
 In-line 58
 Parameters, control statements 7
 PAUSE 39
 Position header 79
 Program check 40
 Program interrupt messages 51-52
 Program size
 Considerations 20
 Object program 21
 Source program 20
 Program status word 51-52,81
 Program transition from 1442 to 2540 24
 Protection check 40
 PSW (See Program status word)

 Register use 54
 Relocation header 79
 REP control statement 28,30-31

RLD statement 78-79

Sample calling subprogram linkage 55
 Sample edit jobs 36-37
 Save area 54-57
 Segment naming
 12-2-9 EDR control statement 32
 12-2-9 REP control statement 30
 Service subprogram descriptions 73-75
 Service subprograms
 DUMP 74
 DVCHK 75
 EXIT 73
 OVERFL 75
 SLITE 74
 /SET control statement 13-16,29-30
 Sharing data in COMMON 58
 SIGN subprogram 73
 SIN subprogram 34,64
 SINGLE option 7-8
 SIZE option 11
 SLITE subprogram 74
 SNGL subprogram 73
 Source program
 Diagnostic messages 20,42-49
 Size 20
 SQRT subprogram 34,61
 Standard options (See also Options) 7
 Starting instructions 40
 Storage address 40
 Subprogram facilities 5
 System maintenance 5
 System tape, FORTRAN 27

T (See Data conversion)
 Tape density 14
 Tape option 17
 TANH subprogram 34,63
 12-2-9 EDR control statement 28,31-33
 12-2-9 END control statement 28,36,79-80
 12-2-9 ESD control statement 76-78
 12-2-9 REP control statement 28,30-31
 12-2-9 RLD control statement 78-79
 12-2-9 TXT control statement 78
 2540 card read punch 24-25
 Types of jobs 40
 TXT statement 78

User-written subprogram names 34

W (See Warning messages)
 Warning messages 42-43,45,48-49



READER'S COMMENTS

IBM System/360 Basic Programming Support
 Title: FORTRAN IV, 360P-FO-031
 Programmer's Guide

Form: C28-6583-0

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?

___ As an introduction to the subject
 Other _____

___ For additional knowledge

fold

Please check the items that describe your position:

___ Customer personnel	___ Operator	___ Sales Representative
___ IBM personnel	___ Programmer	___ Systems Engineer
___ Manager	___ Customer Engineer	___ Trainee
___ Systems Analyst	___ Instructor	Other _____

Please check specific criticism(s), give page number(s), and explain below:

___ Clarification on page(s)
 ___ Addition on page(s)
 ___ Deletion on page(s)
 ___ Error on page(s)

Explanation:

CUT ALONG LINE

fold

Name _____

Address _____

FOLD ON TWO LINES, STAPLE AND MAIL
 No Postage Necessary if Mailed in U.S.A.

fold

fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

CUT ALONG LINE

Printed in U.S.A.

fold

fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York

staple

C28-6583-0